

PATENT ABSTRACTS OF JAPAN

(11) Publication number : 2003-131772

(43) Date of publication of application : 09.05.2003

(51) Int.Cl.
 G06F 3/00
 G06F 3/16
 G10L 15/00
 G10L 15/28

(21) Application number : 2002-131950
 (71) Applicant : MICROSOFT CORP
 (72) Inventor : WANG KUANSAN
 HON HSIAO-WUEN

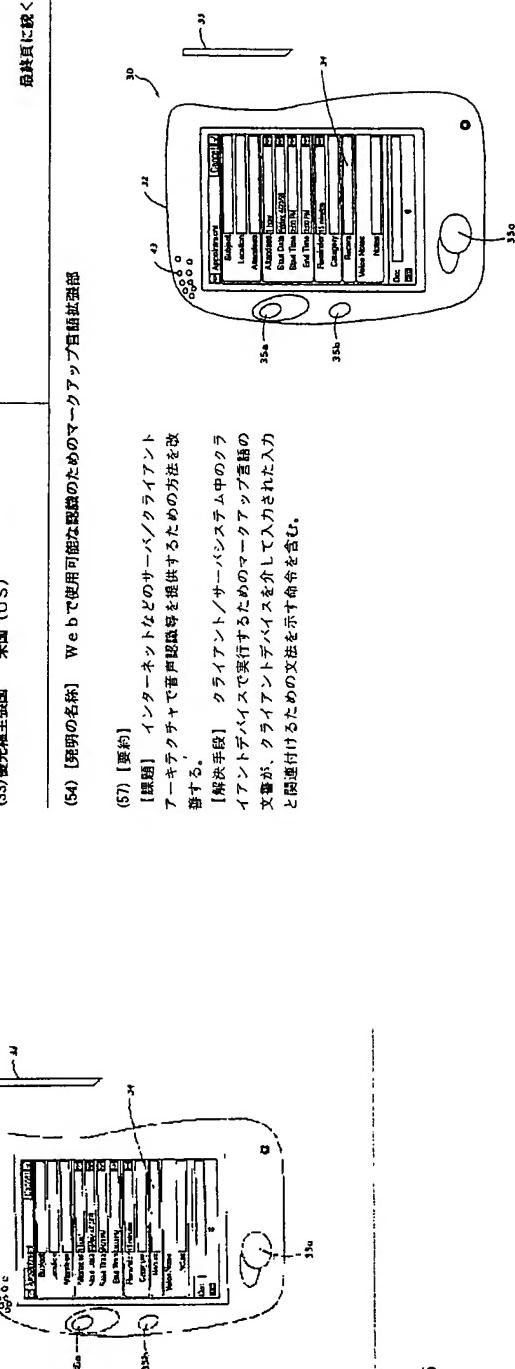
(30) Priority
 Priority number : 2001 289041 Priority date : 04.05.2001 Priority country : US
 2001 960233 20.09.2001 US
 2002 117141 05.04.2002 US

(54) MARKUP LANGUAGE EXTENSIONS FOR RECOGNITION USABLE IN WEB

(57) Abstract:

PROBLEM TO BE SOLVED: To improve a method for providing speech recognition and so on through a client/server architecture such as the Internet and so on.

SOLUTION: Documents of a markup language executed on a client device in a client/server system includes instructions indicating a grammar for association with input entered through the client device.



(19) 日本国特許庁 (JP) (11) 特許出願公開番号 (原文略 < 51 頁)

(P2003-131772
(P2003-131772A)

(43) 公開日 平成15年5月9日 (2003.5.9)

(51) Int.Cl.	F1	F2	F3
G06F 3/00	6 0 1	6 0 1	6 0 1
G06F 3/16	3 2 0	3 2 0	3 2 0
G10L 15/00	15/00	15/28	15/28

(21) 出願番号 特願2002-131950 (P2002-131950)

(22) 出願日 平成14年5月7日 (2002.5.7)

(31) 既先掲主張番号 6 0 / 2 8 9 , 0 4 1	(32) 既先日 平成13年5月4日 (2001.5.4)
(33) 既先掲主張国 米国 (US)	
(31) 既先掲主張番号 0 9 / 9 6 0 , 2 3 3	
(32) 既先日 平成13年9月20日 (2001.9.20)	
(33) 既先掲主張国 米国 (US)	

(24) 代理人 弁理士 谷 篤一 (612名)

(54) [発明の名称] Webで使用可能な認識のためのマークアップ言語拡張部

(57) [要約]
【課題】 インターネットなどのサーバー/クライアントアーキテクチャで音声認知を提供するための方法を改善する。

(57) [解決手段] クライアント/サーバー・システム中のクライアントデバイスで実行するためのマークアップ言語の文書が、クライアントデバイスを介して入力された入力と関連付けるための文法を示す命令を含む。

LEGAL STATUS
 [Date of request for examination] 06.05.2005
 [Date of sending the examiner's decision of rejection]
 [Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]
 [Date of final disposal for application]

特許情報の範囲

請求項 1】 クライアント／サーバシステム中のクラント／サーバで実行するためのマークアップ言語を記載するコンピュータ読取り可能媒体であって、前記マークアップ言語が、クライアントデバイスを介して入力されると同時に、前記マークアップ言語を示す命令を含むこと。

【請求項 1-2】 前記指示が、前記文法の場所の参照を提供することを特徴とする請求項 1-1に記載のコンピュータ読み取り可能媒体。

【請求項 1-3】 前記指示が、音声認識用の音質の参照を含むことを特徴とする請求項 1-2に記載のコンピュータ読み取り可能媒体。

XMLドキュメントを作成することが可能であり、これらのドキュメントは、VoiceXMLインターフリタによってユーザが聞き取れるようになります。ユーザは音声認識を介した音声コマンドを使用して、Webをナビゲートすることができます。

【問題】14 クライアント/サーバネットワークで接続のための方法であって、
の接続のための方法であって、
クライアントデバイスとは異なるネットワーク上のアド
レスにあるWebサーバから、ネットワークに接続され
たクライアントデバイス上に入力データ用のフィールド
を有するマーカップ言語ベースで受け取るステップ
と、

フロー制御タグを備えたマークアップ言語であるが、フロー制御は、イベントスクリプトおよび分離スクリプトを含むHTML(Hyper Text Markup Language)フロー制御モデルに従うものではない。むしろ、Voice XMLは一般に、特に電話ベースの音声専用対話に好適な形態能形アルゴリズムを含み、音声から意味を抽出するためのシーケンサの操作下に置かれる。グラフィカルエディタはアプリケーションインターフェースも提供されているクライアントサーバー構造で使用できるアプリケーションに、Voice XMLを直接組み込むには、開発者が、Voice XML用のものとHTML(または両者のもの)を使用するものであり、それぞれが異なるフローモデルに沿って動作する。

【請求項 15】 前記データを送信する前に、前記入力表示部に示す前記データを標準化するステップをさらに含むことを特徴とする請求項 14 に記載の装置及び方法。

【請求項 16】 誤認識された音声に因る誤認結果を受け取り、前記結果を前記クライアントデバイス上にあらわすデータファイルに貼り付けるステップをさらに含むことを特徴とする請求項 4 に記載の装置及び方法。

【請求項 17】 クライアント/サーバーシステム中のデータファイルに貼り付けるステップをさらに含むことを特徴とする請求項 1 に記載の装置及び方法。

いろいろ、この方法のワーカーショップを自分でやる必要がある。

【0006】したがって、インターネットなどのサーバー／クライアントアーキテクチャで音声認識を提供するのに使用されるアーキテクチャおよび方法を改善することが引き続き求められている。音声認識用のオーサリングツールは、PDAや電話などの小型のコンピューティングデバイスに容易に適応すべきである。前述の欠点のうちの一つ、あるいはすべてに対処するWPSオーサリングアーキテクチャおよび方法が特に求められている。

【0007】**【課題を解決するための手段】** クライアント／サーバーシステム中のクライアントデバイスで実行するためのマー

前記マーカップ書類が、前記テキス
ト文書を処理するために前記文書を閲覧行
ることを特徴とする請求項 9 に記載のコンピュータ読み取り可能媒体。
【請求項 10】 前記要件を特徴とする請求項 1 に記載の文法の指示を含
むことを特徴とする請求項 1 に記載のコンピュータ読み取り可能媒体。
【請求項 20】 前記要件が、要旨などのテキスト文字

クライアントデバイスを介して入力された入力データに関連付けるための文法を示す命令がクライアントデバイスには、クライアントデバイスを介して入力された入力データに他の属性を適用できることで、クライアントデバイスは、入力データ用のフィールドを有する Web サーバからマークアップ言語ページを受け取るために命令を実行することができます。次いでクライアントデバイスは、そのフィールドに開通するユーザーから入力データを受け取り、そのデータおよび接続済のための文法の指示に従って処理します。典型的には処理用にリモート位置に配置された認証サーバへ配達します。

【語訳2-2】 ブラウザソフトウェアで実行するためのマークアップ言語
ライアンデバイスで実行するためのマークアップ言語

- (4) (5) CPU 5.0 と双方間に結合された不揮発性記憶リード / 書込
 メモリ (RAM) 5.0 が搭載される。

[0.01.0] [説明の実施の形態] 同じ内容を英語するための W eb ベースの認識およびソフトウェアのアーキテクチャについて述べる前に、そのアーキテクチャについて説明するが、一般的なコンピュータディバイスについて説明する方が有りである。図 1 を参照すると、データ管理デバイス (PIM, PDA など) の形態例が 3.0 に示されている。ただし、本発明は、以下で論じる他のコンピュータディバイス、具体的に言えば、入力 / 出力ボタン用などの要素の RAM 5.4 へのローディングを制御する、デバイス用のオペレーティングシステムソフトウェアを格納して隠されている。したがって、本発明から電話を受けることができる。したがって、本発明は、既存の機器を受けることになる。こうしたデバイスは、既存の機器用個人情報管理デバイスおよび他の機器用電子デバイスに比べて、機能強化されたユーティリティを有するものとなり、こうしたデバイスの機能およびコンピュータ管理または PIM デバイス、電話、またはコンピュータの間連により、ユーザーに対して常に待ち逗留するように働きやすくなる。したがって、本明細書に記載されたデータ管理または PIM デバイス、電話、またはコンピュータの顯示によって選択されることを意図するものではない。

[0.01.1] データ管理移動デバイス 3.0 の例示的形態が、図 1 に示されている。移動デバイス 3.0 はハブシングル 3.2 を含み、スタイルラス 3.3 と一緒に搭載する接続端子の面版に表示するドライブスワイプ 3.4 を含むユーザインターフェースを使用する。スタイルラス 3.3 は、フィールドを選択するために指定された座標位置でディスプレイ 3.4 を押すかまたは接触するため、カーソルの位置位置を選択的に移動するため、あるいはその他の方法でジェスチャまたは手書きなどのコマンド情報を提供するため、使用される。別例では、または加え、ディスプレイ 3.0 上にナビゲーション 1.1 または複数のボタン 3.5 を含むことができる。さらに、回転ボール、ローラなどの他の入力メカニズムを提供するこどもできる。ただし、本発明は、これらの入力メカニズム形態によって限定されるものではないことに留意されたい。たとえば他の入力形態には、コンピュータビジョンを介するなどの規定的な入力を含むことができる。

[0.01.2] 次に図 2 を参照すると、移動デバイス 3.0 を含む機器構成要素が示されている。中央処理ユニット (CPU) 5.0 は、ソフトウェア制御機能を実施する。CPU 5.0 がディスプレイ 3.4 に結合されているため、制御ソフトウェアに從つて生成されたテキストおよびグラフィックアイコンがディスプレイ 3.4 に表示される。スピーカ 4.3 は、可聴出力を提供するために、典型的にはデジタルオーディオ装置 (DA) 5.0 を使って C インターフェース 3.0 に入力される。スピーカ 4.3 は、データはハードドライブ 3.0 に入力されると、音声データが、データはハードドライブ 3.0 に入力されると、音声データが、データは

[0.01.3] RAM 5.4 は、アプリケーションプログラムを構成するのに使用される PC 上のハードドライブの機能に類似した方法で、コード用の記憶域としての役割も果たす。コードを格納するには不揮発性メモリが使用されるが、代替方法として、コードの実行には使用されない揮発性メモリ内に格納することもできる。したがって、本発明は、既存の機器を受けることによって送受信トランザクション 5.2 を介して移動デバイスによって送受信が可能である。コンピュータ (たとえばデスクトップコンピュータ) から、または壁むすら接続ネットワークから、データを直接ダウンロードするために、オブジョンの通信インターフェース 6.0 を提供することもできる。したがってインターフェース 6.0 は、たとえば、赤外線リンク、ネットワークカードなどの様々な外形の通信デバイスを含むことができる。

[0.01.4] 無線通信は、CPU 5.0 に結合された無線トランザクション 5.2 を介して送受信が可能である。コンピュータ (たとえばデスクトップコンピュータ) から、または壁むすら接続ネットワークから、データを直接ダウンロードするために、オブジョンの通信インターフェース 6.0 を提供することもできる。赤外線リンク、ネットワークカードなどの様々な外形の通信デバイスを含むことができる。

[0.01.5] 移動デバイス 3.0 は、マイクロフォン 2.9 およびアナログデジタル (A/D) 変換器 3.7、ならびに記憶装置 5.4 に格納されたオーバージョンの認識プログラム (音声、DTMF、手書き、ジェスチャ、またはコンピュータビジョン) を含む。例を挙げると、ディスプレイ 3.0 のユーザの操作によるナビゲーション 1.1 またはカーソルの位置位置を正規化するため、正規化および/またはマイクロフォン 2.9 が音声信号を取得し、これが A/D 変換器 3.7 によってデジタル化される。音声認識プログラムは、音声認識中間結果を取得するために、デジタル化された音声信号に対して、音声データ (または無効出力機能を実行することができる。音声データは、マイクロフォン 2.9 または音声インターフェース 4.0 は、無線トランザクション 5.2 または通信インターフェース 6.0 を使用して、以下で考証した図 5 のアーキテクチャに示される。リモート認證サーバ 2.0 に伝送される。その後認証結果は、提示し (たとえば投票のおよび/または感覚的) 、かつ結果として W eb サーバ 2.0 (図 5) に伝送するために移動デバイス 3.0 に戻され、W eb サーバ 2.0 および移動デバイス 3.0 はクライアント / サーバの関係で動作する。同様の処理が、他の入力形態に適用される。たとえば手書き入力が、データは上での取扱いによることが可能である。一般に、プログラムモジュールには、特定のタスクを実行するか、

[0.01.6] 図 3 は、携帯電話 8.0 の例示的実施形態を示す平面図である。電話 8.0 およびキーパッド 8.4 が含まれる。一般に図 2 の構成要素 3 の電話に適用されるが、他の機能を実行するのに必要な追加の回路を要求することができる。たとえば、図 2 の実施形態では、電話として動作することが必要なトラシングが要求されるが、こうした回路は本発明には関係していない。

[0.01.7] 前述の機器または移動コンピュータ / イングデバイスに加えて、本発明が、汎用デスクトップコンピュータなどの他の多數のコンピューティングデバイスで使用可能であることを理解させたい。たとえば本発明は、身体的能力の制限されたユーザーが、フル英数字キーボードなどの他の花形型入力デバイスを操作するのが困難である場合、テキストをコンピュータまたは他のコンピュータ / イングデバイスに入力できるようにするものである。

[0.01.8] 本発明は、他の多數の汎用または専用用途向けコンピューティングシステム、環境、または構成で最も動作可能である。本発明で使用するのに好適な、よく知られたコンピューティングシステム、環境、および/または構成の例には、通常の電話 (画面なし)、ハーネスナルコンピュータ、サーバコンピュータ、ハンドヘルドまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースシステム、セットトップボックス、プログラム可能な大容量電子製品、ネットワーク PC、ミニコンピュータ、メインフレームコンピュータ、上記システムまたはデバイスのいずれかを含む分離されたものではない。

[0.01.9] 次に、図 4 に示した汎用コンピュータ 1.2 0について、簡単に説明する。ただし、コンピュータ 1.2 0 は、好適なコンピューティング環境の一例に過ぎず、本発明の使用または機能の範囲に於て何らかの制限を示唆するものではない。さらにもコンピュータ 1.2 0 は、本明細書に開示された構成要素のいずれか 1 つまたはそれ以上の組合せに関して、なんらかの依存性または要求条件を有するものとして解釈されるものでない。

[0.01.10] 本発明は、コンピュータによって実行されるプログラムモジュールなどの、コンピュータ実行可能な命令の一般的な文脈で説明することができる。一般的に、プログラムモジュールには、特定のタスクを実行するか、

-

(11)

19

スタイルス3.3を使用してフィールド2.5.0を選択する
と、イベント「On Click」が起動され、これがス
クリプト部分2.7で関数「(a)k」を呼び出すま
たは実行する。この動作により、通常はフィールド2.5
で予想されるデータのタイプに超過付けられた音声認
識に使用される文法が起動する。このタイプの文法が
「Multimedia」と呼ばれ、複数の入力方法
(たとえば音声およびクリック／ローラー) が含まれ
る。

【0.4.6】多くのアプリケーションで、ソースページ
アブリケーション指定のグラフィック機能を使用する
ことによって、作者がページの様々な構成要素を音声化
できるようにする機能を実現するため、図8
に示された音声認識部は、クラウントアントのラグラ
ム上でデフォルト時に選択状況を表示するものでは
ないことに留意された。それにもかかわらず、復
元表現が選択される場合は、拡張部をそのように修正する
ことができる。

【0.4.7】再度文生に戻ると、文法は、文庫自由文
法、N文法、またはハイブリッド文法などによって生
じる可能性がある。もちろん、対応する認識形態が使
用されないと、構文文法である。(もちろん、対応する
認識形態が選択されると、DTMF文法、手
書き文法、ジェスチャ文法、およびイメージ文法が使
用される。) 本明細書では、「文法」には、認識を実行す
るための情報が含まれ、他の実施形態では、たとえば特
定のフィールドに入力されると予想される入力に対する
情報が含まれる。マークアップ文書の第1の強調部を
示す新しい制御2.9.0(本明細書では「reco」とし
て識別)には様々な要素が含まれるが、そのうちの2
つ、すなわち文法要素「grammar」および「bi
nd」要素が図示されている。一般に、Webサーバー2
0.2からクラウントアントにダウンロードされるコードと同
様に、文法はWebサーバー0.2側から発信される。クラ
ウントアントはダウンロードされるか、あるいは
APIによってインラインまたは参照済み
文書のためリモートサーバーに転送されることが可
能である。その後文生は、そこでキャッシュにローカル
に格納することができる。最終的に文法は、認識特に使
用するために認識サーバー2.0.4に送られる。grammar
要素は、属性を使用するインラインまたは参照済み
のいずれか文法を指定するのに使用される。たとえば
【0.4.8】認識された音声、手書き、ジェスチャ、イ
メージなどに対応する認識サーバー2.0.4からの認識結果
を受け取る直後に、reco制御2.9.0の例が、対
応する結果を受け取り、対応するフィールドに超過付け
るように提供され、フィールドは、ディスプレイ3.4上
でその中のテキストを表示することを含むことができ
る。例示された実施形態では、結果が超えたときに超
したときに表示される。この結果は、クライアント3.0で記
録され、認識サーバー2.0.4に送られる音声データを伴う

特開2003-131772

18

し、各フィールド2.5.2および2.5.4に一意の「reco
オブジェクトおよび文法が選択され、認識されたテキス
トを受け取ると、フィールド2.5.2および2.5.4それぞ
れに超過付けられる。カード番号フィールド2.5の受
取りに関しては、图7に示しておいたのと同様
の方法で、閑散「(handel)」がカードタイプを基準
にカード番号の値をチェックする。

【0.4.9】一般に、アーキテクチャ2.0.0およびグラ
ムアント側マークアップ言語と共に音声認識を使用する
場合、第1に、与えられた音声に閑適付けられたフィー
ルドが示される。例示された実施形態では、スタートス
3.3が使用されるが、本発明は、スタイルス3.3の使用
に限らず、ボタン、マウスボタン、マウスandler、
回転がイーラなどの任意の形態の指示が使用できること
を理解されたい。【On Click】などのコマンドの
使用による音声を検出し、2.8.7で音声が終
了したことを特定した場所を示すものである。

【0.5.4】Reco要素の様々な属性は、認識サーバー
2.0.4が発行されるが、この動作モードで
3.0.5以前の認識できない会話によって生成された「o
nRecognition」イベント2.9.3は、認識を停止しな
い。ただし、「bubbleTimeout」属性2.9
または「maxLength」属性2.9.9に超えて
2.9.1が発行される。認識サーバー2.0.4が認識でき
ない会話を超えると、認識が停止する。

【0.5.5】图1.4は、認識サーバー2.0.4の「multi
pileMode」動作を示す会話図である。前述の
ように、この動作モードは「openMicrograph
one」属性または「bubbleTimeout」属性2.9.1—コール
3.0.5が受け取られるが、または「bubbleTimeout
on」属性2.9.5または「maxLength」属性2.9.9に
超えて2.9.1が発行される。認識サーバー2.0.4が
2.9.1が認識できない会話を超えると、エラーが生じたかどうかに
応じて、異なるイベントが発行される。認識サーバー2.0
4が依然として音声を処理している場合、たとえば非常
緊急の電話の場合、「onNoRecognition」属性2.9.3が
発行される。ただし、任意の他の理由で「bubble
Timeout」属性2.9.5を超えて2.9.1が認識新規の
エラーである可能性が高く、「onTimeout」イベント2.9.7が発行
される。認識を停止している場合、たとえば音
声を超過するグラフ上で実行中のプログラムによって
音声の先頭となるフィールドを示すことができるこ
とで、音声を認識されたい。

【0.5.6】ここで、様々な音声認識ソリオガ、認識
を停止または取り消すことができる他の
属性に「bubbleTimeout」属性2.9.5が
含まれるが、これは認識サーバー2.0.4が2.8.5で音声を
検出した後に結果を戻さなければならぬ時間枠であ
る。この時間枠を超えると、エラーが生じたかどうかに
応じて、異なるイベントが発行される。認識サーバー2.0
4が依然として音声を処理している場合、「onNoTime
out」属性2.9.9も提供することが可能であり、これ
は認識の開始2.8.3から結果がグラフ上に戻さ
れるまでの時間枠に関するものである。この時間枠を超
えると、「onTimeout」イベント2.9.7が発行
される。

【0.5.7】前述の例では、マルチモーダルアリケー
ションのユーザーは、たとえば、圧力感知ディスプレイ上
で覗きき保持することによって、デバイスへの入力が
認識される。次いでグラフ上に表示され
ばならない時間枠を削除し、認識結果を戻す。だ
けでなく、電話アリケーション(以下で考案)またはハン
ズフリーアリケーションなどの音声専用ソリオでは、
ユーザーがグラフ上に表示されることがないた
め、認識サーバー2.0.4またはグラフィックアリケー
ションのために認識結果を削除する。ただし、
認識結果を削除する場合、認識が完了した
ことを明示するため、認識サーバー2.0.4が認識を
自動的に停止して、その結果を戻す。認識サーバー2.0.4
は、認識結果を戻すべきであるかどうかを判定するた
めの確度測定(confidence measure)50

(10)

と、イベント「On Click」が起動され、これがス
クリプト部分2.7で関数「(a)k」を呼び出すま
たは実行する。この動作により、通常はフィールド2.5
で予想されるデータのタイプに超過付けられた音声認
識に使用される文法が起動する。このタイプの文法が
「Multimedia」と呼ばれ、複数の入力方法
(たとえば音声およびクリック／ローラー) が含まれ
る。

【0.4.6】多くのアプリケーションで、ソースページ
アブリケーション指定のグラフィック機能を使用する
ことによって、作者がページの様々な構成要素を音声化
できるようにする機能を実現するため、図8
に示された音声認識部は、クラウントアントのラグラ
ム上でデフォルト時に選択状況を表示するものでは
ないことに留意された。それにもかかわらず、復
元表現が選択される場合は、拡張部をそのように修正する
ことができる。

【0.4.7】再度文生に戻ると、文法は、文庫自由文
法、N文法、またはハイブリッド文法などによって生
じる可能性がある。もちろん、対応する認識形態が使
用されないと、構文文法である。(もちろん、対応する
認識形態が選択されると、DTMF文法、手
書き文法、ジェスチャ文法、およびイメージ文法が使
用される。) 本明細書では、「文法」には、認識を実行す
るための情報が含まれ、他の実施形態では、たとえば特
定のフィールドに入力されると予想される入力に対する
情報が含まれる。マークアップ文書の第1の強調部を
示す新しい制御2.9.0(本明細書では「reco」とし
て識別)には様々な要素が含まれるが、そのうちの2
つ、すなわち文法要素「grammar」および「bi
nd」要素が図示されている。一般に、Webサーバー2
0.2からクラウントアントにダウンロードされるコードと同
様に、文法はWebサーバー0.2側から発信される。クラ
ウントアントはダウンロードされるか、あるいは
APIによってインラインまたは参照済み
文書のためリモートサーバーに転送されることが可
能である。その後文生は、そこでキャッシュにローカル
に格納することができる。最終的に文法は、認識特に使
用するために認識サーバー2.0.4に送られる。grammar
要素は、属性を使用するインラインまたは参照済み
のいずれか文法を指定するのに使用される。たとえば
【0.4.8】認識された音声、手書き、ジェスチャ、イ
メージなどに対応する認識サーバー2.0.4からの認識結果
を受け取る直後に、reco制御2.9.0の例が、対
応する結果を受け取り、対応するフィールドに超過付け
るように提供され、フィールドは、ディスプレイ3.4上
でその中のテキストを表示することを含むことができ
る。例示された実施形態では、結果が超えたときに超
したときに表示される。この結果は、クライアント3.0で記
録され、認識サーバー2.0.4に送られる音声データを伴う

を実施できることに留意されたい。認識結果が大きい値
よりも低い場合は「onRecognition」属性2.9.3が発
行され、確度測定がしきい値よりも高い場合は「onR
ecognition」属性3.0.3が発行される。図1
4は、「Automatic mode」で、明示的な
stop()コールが実行されていないことを示す。
【0.5.7】图1.5は、認識サーバー2.0.4の「sing
le mode」動作を示す会話図である。「auto
matic mode」属性に關して上記述べた属性およ
びイベントが適用可能である。その場合は同じ參照号
が自動化されない場合の動作モードでは、stop()
で示されている。ただし、この動作モードでは、stop()
コール3.0.5が時間間隔2.8.1上に示されている。
stop()コール3.0.5は、ユーザによる「pen
up」などのイベントに対応するものである。この動作
モードでは、認識結果を戻すことが、明示的なstop()
()コール3.0.5の削除下にあります。すべての動作モード
と同様に、音声が「initialTimeout」時間
間隔2.8.9内に検出されない場合は「onSilenc
e」イベント2.9.1が発行されるが、この動作モードで
は、認識が停止されない。同様に、stop()コール
3.0.5以前の認識できない会話によって生成された「o
nRecognition」イベント2.9.3は、認識を停止しな
い。ただし、「bubbleTimeout」属性2.9
または「maxLength」属性2.9.9に超えて
2.9.1が発行される。認識を超えた時間枠を超
えて2.8.5で音声が検出し、2.8.7で音声が終
了したことを特定した場所を示すものである。
【0.5.4】Reco要素の様々な属性は、認識サーバー
2.0.4が発行されるが、この動作モードで
3.0.5以前の認識できない会話によって生成された「o
nRecognition」イベント2.9.3は、認識を停止しな
い。ただし、「bubbleTimeout」属性2.9.3は、認識を停止しな
い。ただし、「bubbleTimeout」属性2.9.5または「maxLength」属性2.9.9に超えて
2.9.1が発行される。認識サーバー2.0.4が認識でき
ない会話を超えると、認識が停止する。
【0.5.8】图1.6は、認識サーバー2.0.4の「multi
pileMode」動作を示す会話図である。前述の
ように、この動作モードは口述シナリオで使用される。一般
の場合は口述シナリオで使用される。この動作モードでは、明示的なstop()—コール
3.0.5が受け取られるが、または「bubbleTimeout
on」属性2.9.5または「maxLength」属性2.9.9まで
認識結果が戻される。ただし、認識を停止しない任の
「onSilence」イベント2.9.1が発行される。ただし、認識を停止
したことでも留意されたい。
【0.5.9】图1.7は、認識サーバー2.0.4の「multi
pileMode」動作を示す会話図である。前述の
ように、この動作モードは口述シナリオで使用される。一般
の場合は口述シナリオで使用される。この動作モードでは、明示的なstop()—コール
3.0.5が受け取られるが、または「bubbleTimeout
on」属性2.9.5または「maxLength」属性2.9.9まで
認識結果が戻される。この時間枠を超えるまで、時々
認識結果が戻される。ただし、認識を停止しない任の
「onSilence」イベント2.9.1が発行される。ただし、認識された
オペレーターが認識結果を戻す。ただし、「onNoRec
ognition」属性3.0.3または「onNoReco」イ
ベント2.9.3の後、「bubbleTimeout」時間
枠および「maxLength」属性2.9.9に超えて2.9.1が
発行される。認識できない状態により「onSil
ence」イベント2.9.1が発行される場合、これらのイベ
ントは報告されるが、認識は継続する。
【0.6.0】前述のように、フィールドの開閉付けら
れた認識結果が戻された場合は、認識が完了した
ことを明示するのであり、認識サーバー2.0.4が発行さ
れた「recoオブジェクト」が起動され、これには少なく
とも、文法を使用する認識サーバー2.0.4の指示を提供す
ることが含まれる。この文法は、クライアント3.0で記
録され、認識サーバー2.0.4に送られる音声データを伴う

2) すべてのプロンプトおよび新しいオブジェクトを起動する
前述のように、音声データは、ユーザが
音声に聞かせられた音声で入力されたストリーミングデータを
示すことを可能にする。しかし、文法は、図8に示して
全体的に抑制される。また、音声データは、音声
上記述べたものと同じ文脈を使用して起動され、音声
データおよび使用する文法の指示が認識サーバ204に
提供される。同時に、認識サーバ204から受け取った
出力は、クライアントのファイルドライブに直接受けられる
(本明細書では電話音声プログラマ212)。

【0064】一般に音声専用アプリケーションに固有の
他の特徴は、音声が認識されなかつたときのユーザーに対
する指示である。図8などのマルチモーダルアプリケー
10

10
のめの加重部がさらに2つある。DTMFはreco制御
にマッピングする出一の文法を指定する。たとえば、
「1」は料金録音部門、「2」は累積部門を意味するな
どである。これに対して、コードオブジェクトは、呼出し
送達および第三者に対する電話機能を処理する。風
景、性、プロトタイプ、メッシュ、およびイベントについて
は、付録で詳細に論じる。

10.0.6.1 録音サーバ 2.0 から認証結果を受け取ることを示す際に表示されたフィールドに nul1 が入るだけであるため、それ以上の動作は必要ない。音声用の要適応形では、「fon Reco」 3.0 が開発されたまでは実行し、録音サーバ 2.0 が開発されたまでは実行し、これが好適なディスクストラムへの変換システム 3.0.7 (図5) を使用して、音声に変換される。録音サーバ 2.0 は可聴ストリームを音信号ブロック 2.1.2 に展し、次にこれが電話 8.0 に伝送されユーザが聞くことになる。同様に、音声データはアリケーションの他の形態プロンプトと一緒に記録される。2.0 によって可聴ストリームに変換される。

されれば、システムはユーザの話を開始させるか、そこで話題を開始するようになります。これがユーザが話題を提供する形態であればユーザが話題を開始することができる。これが「主導権混流（mixed initiative）」の一例である。一般にこの形態の対話（dialog interaction）では、ユーザが対話の主導権をシステムと共有することができる。ユーザがボタンを押すことで情報を提供する、前述および以下で詳細に論じる例に加えて、ユーザは指示されていない場合にもタスクを切り換えることができる。

してウェルムプロンプトを再生した後、関数 `l i c h e ck F i l l e d` がユーザーに対して各フィールドについてのプロンプトを出し、入力されたフィールドの反復および情報が正しいことの確認を含む適切な文法を起動するのである。これでは「`i n f o r m a t i o n`」が起動される場合が含まれることに留意されたい。この実現では、それぞれの `r e c o` 開頭が、以前の例の本 `n` 文法の起動が含まれることに留意されたい。

10 0.6.2 図 8 に示された実現形態には図示していないが、`r e c o` 開頭には、適切な音声データを認識するためのリモート可聴オブジェクト `R A O` を含むことでもできる。RAO をブライアン `b r a y e n` が `2.0.4` に向けて送るための、リモート可聴オブジェクト `R A O` を含むことでもできる。

11 0.6.2.1 図 8 に示された実現形態には図示していないが、`r e c o` 開頭には、適切な音声データを認識するためのリモート可聴オブジェクト `R A O` を含むことでもできる。RAO をブライアン `b r a y e n` が `2.0.4` に向けて送るための、リモート可聴オブジェクト `R A O` を含むことでもできる。音声入力ベースのクラウドおよびディスクлейなし、音声入力ベースのクラウドおよびディスクлейなし、音声入力ベースのクラウドでは、それが `r e c o` 開頭が、以前の例の本 `n` 文法の起動が含まれることに留意されたい。

— types], 「g_cardnum」、および
「g_expirydate」に随選された情報を
が含まれる。この例では、「inRecd」で示すと
は既報された音声を受け取る、電話番号ラウザ
2212が電話800から受け取った音声データおよび文法
「dofield」を使用するための指示を認証サー
バ204に送信し、音声データからの認識された任务ま
たはすべてのフィールドに随選する値を随選けることを
含むも関係「handle」が呼び出されるとからは実
行される。言い換えれば、認証サーバ204から取得し
た結果に22、各フィールドに随選する指示も含まれるとい
うことである。この情報が解析され、405に指定され

40 41

特開2003-131772
26

(14)

25
ltimeida IntegrationLangauge
a ge (SMIL)、および近日発表されるW3C X
ML(マークアップ言語)要素を含むが、これらに限定されるも
のではない。他のインティンク機構に、<b ind>
の用法を一概にするのは簡単であることを理解されると
とに留意されたい。

[0076] 図17および図18は、クライアント、具
体的にはd o w n l e v e l b r o w s e r上で実行
ファイル(d r i n k)に認証結果の値が割り当てら
れ、次いでユーザは、3 7 4で、プロンプトオブジェクト
[s k i p]が、3 6 8でプロンプトオブジェクト [r e c o _ d
**rink]が実行され、3 7 0で認証オブジェクト [r e c o _ d
rink]が再開される。戻された認証結果が、確認値**

が1 0よりも大きい[co f f ee]の場合、3 7 2で
属性は[on R e c o]である。

[008 1] 高位要素としての[b ind]要素は、付
録2、4項に指定されたいずれのイベントも含むことを
ができる。さらに、[b ind]要素は、アクセス可能であ
る場合を希望するか否かに関するプロンプトを受け取る。
次いで3 7 6で、セッション3 5 6の認証オブジェクト
[r e c o _ c r e a m _ s u g a r]が呼び出され
る。これとは違い、認証結果の確認値が1 0よりも大き
いがco f f eeでない場合、3 7 8で再度dr in k
オブジェクトが割り当てられる。3 8 0でプロンプトオブ
ジェクト[co n t a i n e r]を実行し、その後セクション
3 5 2、およびセクション3 5 0、音声セクション3
4、および3 5 6、3 5 8が含まれる。セクション3 5
4は、ユーザの希望する飲み物に関する一般問合せから
4度にわたって飲み物について、対話型認証ア
クションまたは認証結果の確認値で[y e s
no]と答えると、3 8 4でプロンプトオブジェクト [t h a
n k s]が再生され、その後3 8 6で番号がサブミット
される。これとは違い、ユーザが[no]と答えるか、
または認証結果の確認値が1 0よりも大きい場合、3 9 0
でプロンプトオブジェクト [r e t r y]が実行され、
もしくは一度3 9 2でプロンプトオブジェクト [s k i p]が
**実行され、3 9 4で認証オブジェクト [r e c o _ d
rink]が呼び出される。**

[008 1] 頃の所からわかるように、[b ind]

要素は、セッション3 5 8は、飲み物の確認に関する認証結
果を受け取る。セクション3 6 0は新しいメッセージシ
ンオブジェクト[SMEX]を使用する呼出しセクション
であり、これについては以下で詳細に論じる。

[0079]前述のように、本発明の特徴の[bi

nd]要素はオブジェクトメント呼出しが含まれ、
これが3 6 1で[w e l c o m e]オブジェクトの[s
t a r i]メソッドが実行されたときに、「w e l c
**o m e」プロンプトを再生することによって、図17およ
び図18の所でユーザ対話を開始させる。**

[008 1]次にユーザは、3 6 2で[sa k e d]オ
ブジェクトの[start]メソッドを実行することによ
つて、認証が実行される。

[008 0]次にセッション3 5 4のマークアップが実
行され、認証サーバ2 0 4が使用する文字が、Xpa
l hステートメント[. /dr in k t y p e s]によ
り、受け取った認証結果がXpa l h言語
を提供される。この例では、W3CのXpa l h言語
を使用しているが、当分の技術者であれば、W3C
XML黙示規則(XQ L)を含むがこれに限定されるこ
との他の規則も可能である。たとえば、以下のマークアップ

要素は、セッション3 5 6に示すように、所望であ
れば、認証された結果の確認値を宣言することもで
**きる。所示された実施形態では、複数の割当およびメ
ソッド呼出しが宣言されると、ドキュメント順で実行さ
れる。**

[008 2]他の実施形態では、メソッド引数を渡すた
めの規定も提供される。言い換れば、メソッドの中に
は引数リストが必要なものもある。これは、「ra
**ge」サブ要素を使用して達成される。たとえば、以下のマー
クアップ**

[008 3]「star i」メソッドを呼び出すことに
よって、認証が実行される。

[008 0]次にセッション3 5 4のマークアップが実
行され、認証サーバ2 0 4が使用する文字が、Xpa
l hステートメント[. /dr in k t y p e s]によ
り、受け取った認証結果がXpa l h言語
を提供される。この例では、W3CのXpa l h言語
を使用しているが、当分の技術者であれば、W3C
XML黙示規則(XQ L)を含むがこれに限定されるこ
との他の規則も可能である。たとえば、以下のマークアップ

特開2003-131772
27

(15)

'prompt' targetMethod = "start"/>
 是、[on N o R e c o]イベントが送られたときを意味するものであり、オブジェクト['pr o p r o m p t']の[start]メソッドが呼び出されたように、「b ind】要素を[R e c o]要素の子要素として使用することによって、プロンプト要素を使用して動的コンテンツまたは可読性ファイルにテキストメッセージを提供することが可能である。インベーティング認証結果に關するイベントと同様とするとことができる。たとえば、インベーティングと[on R e c e i v e d]を含むことが可能である。メッセージソース(たとえばクライアントデバイス上で実行中のアプリケーション)がブラウザにメッセージを有する場合に送られる。

[008 7]したがって「s m e x」またはメッセージオブジェクトは、本明細書で論じるマークアップタグを、クライアントデバイス上で実行中の他の構成要素まではも同様にアクセスできる。

[008 5] 認証結果の条件をチェックすることに加えて、実行予定の現在のドキュメントまたはページもチェックすることができる。具体的には、「t e s t」属性および「v a l u e」属性の両方を、格納ドキュメントのルートノードを示す「h o s t」プリミティブをある他の例と同様にアクセスする。その後このメッセージは、クライアントデバイス上で認証結果が受け取られた場合と同様に使用され、メッセージが解析されて認証のフィールドに割り当てられると、または前述の「r e c o m m a n d」、または「b ind」などの属性を使用する代わりに、ユーザが入力した内容のメッセージを提供する。その後このメッセージは、認証サーバから認証結果が受け取られた場合は、同様に使用され、「s m e x」オブジェクトについては、代價として認証結果が受け取られた場合と同様に解析される。

[008 8]「b ind」要素は、「f o r t」属性を含むこと可能であり、これによってその動作をページ上の他のオブジェクトに結びつけることができる。たとえば、以下のようないわゆるAと同様に解説する。

[008 6]また、「b ind】要素は音声サーバ2 0 4からの認証結果およびドキュメント内部の値の受取りまたは割り当てで適用可能なだけでなく、たとえば、クラウド電話[dr in k f i r e l o g]が「c o n f i c e」により、この属性は音声サーバ2 0 4上で実行中のアブリケーションからの認証結果が実行される。セッション3 5 6では、認証結果が呼び出されたときに、ページは、メッセージオブジェクト(本明細書で示されている)にも適用可能であることにによって認証を開始する。音声サーバ2 0 4から受け取る認証結果と同様に、受け取るメッセージも多種多様である。メッセージ

オブジェクトを実行するために音声サーバ2 0 4から受け取る認証結果と同様に、受け取るメッセージも多種多様である。メッセージ

オブジェクトを実行するために音声サーバ2 0 4から受け取る認証結果と同様に、受け取るメッセージも多種多様である。メッセージ

特開2003-131772
28

は、「O B J . F (X, Y)」と呼称されるが、あるいは、「O B J 」が、「ラムダータまたは引数「X」および「Y」を使用するオブジェクト

である。

[008 3]「b ind】要素は、b ind要素がどのイベント用であるかを直感する「e v e n t」属性を含むことも可能である。たとえば、以下のマークアップ

は、「O B J <arg>X</arg><arg>Y</arg></b ind>

40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
 100
 101
 102
 103
 104
 105
 106
 107
 108
 109
 110
 111
 112
 113
 114
 115
 116
 117
 118
 119
 120
 121
 122
 123
 124
 125
 126
 127
 128
 129
 130
 131
 132
 133
 134
 135
 136
 137
 138
 139
 140
 141
 142
 143
 144
 145
 146
 147
 148
 149
 150
 151
 152
 153
 154
 155
 156
 157
 158
 159
 160
 161
 162
 163
 164
 165
 166
 167
 168
 169
 170
 171
 172
 173
 174
 175
 176
 177
 178
 179
 180
 181
 182
 183
 184
 185
 186
 187
 188
 189
 190
 191
 192
 193
 194
 195
 196
 197
 198
 199
 200
 201
 202
 203
 204
 205
 206
 207
 208
 209
 210
 211
 212
 213
 214
 215
 216
 217
 218
 219
 220
 221
 222
 223
 224
 225
 226
 227
 228
 229
 230
 231
 232
 233
 234
 235
 236
 237
 238
 239
 240
 241
 242
 243
 244
 245
 246
 247
 248
 249
 250
 251
 252
 253
 254
 255
 256
 257
 258
 259
 260
 261
 262
 263
 264
 265
 266
 267
 268
 269
 270
 271
 272
 273
 274
 275
 276
 277
 278
 279
 280
 281
 282
 283
 284
 285
 286
 287
 288
 289
 290
 291
 292
 293
 294
 295
 296
 297
 298
 299
 300
 301
 302
 303
 304
 305
 306
 307
 308
 309
 310
 311
 312
 313
 314
 315
 316
 317
 318
 319
 320
 321
 322
 323
 324
 325
 326
 327
 328
 329
 330
 331
 332
 333
 334
 335
 336
 337
 338
 339
 340
 341
 342
 343
 344
 345
 346
 347
 348
 349
 350
 351
 352
 353
 354
 355
 356
 357
 358
 359
 360
 361
 362
 363
 364
 365
 366
 367
 368
 369
 370
 371
 372
 373
 374
 375
 376
 377
 378
 379
 380
 381
 382
 383
 384
 385
 386
 387
 388
 389
 390
 391
 392
 393
 394
 395
 396
 397
 398
 399
 400
 401
 402
 403
 404
 405
 406
 407
 408
 409
 410
 411
 412
 413
 414
 415
 416
 417
 418
 419
 420
 421
 422
 423
 424
 425
 426
 427
 428
 429
 430
 431
 432
 433
 434
 435
 436
 437
 438
 439
 440
 441
 442
 443
 444
 445
 446
 447
 448
 449
 450
 451
 452
 453
 454
 455
 456
 457
 458
 459
 460
 461
 462
 463
 464
 465
 466
 467
 468
 469
 470
 471
 472
 473
 474
 475
 476
 477
 478
 479
 480
 481
 482
 483
 484
 485
 486
 487
 488
 489
 490
 491
 492
 493
 494
 495
 496
 497
 498
 499
 500
 501
 502
 503
 504
 505
 506
 507
 508
 509
 510
 511
 512
 513
 514
 515
 516
 517
 518
 519
 520
 521
 522
 523
 524
 525
 526
 527
 528
 529
 530
 531
 532
 533
 534
 535
 536
 537
 538
 539
 540
 541
 542
 543
 544
 545
 546
 547
 548
 549
 550
 551
 552
 553
 554
 555
 556
 557
 558
 559
 560
 561
 562
 563
 564
 565
 566
 567
 568
 569
 570
 571
 572
 573
 574
 575
 576
 577
 578
 579
 580
 581
 582
 583
 584
 585
 586
 587
 588
 589
 590
 591
 592
 593
 594
 595
 596
 597
 598
 599
 600
 601
 602
 603
 604
 605
 606
 607
 608
 609
 610
 611
 612
 613
 614
 615
 616
 617
 618
 619
 620
 621
 622
 623
 624
 625
 626
 627
 628
 629
 630
 631
 632
 633
 634
 635
 636
 637
 638
 639
 640
 641
 642
 643
 644
 645
 646
 647
 648
 649
 650
 651
 652
 653
 654
 655
 656
 657
 658
 659
 660
 661
 662
 663
 664
 665
 666
 667
 668
 669
 670
 671
 672
 673
 674
 675
 676
 677
 678
 679
 680
 681
 682
 683
 684
 685
 686
 687
 688
 689
 690
 691
 692
 693
 694
 695
 696
 697
 698
 699
 700
 701
 702
 703
 704
 705
 706
 707
 708
 709
 710
 711
 712
 713
 714
 715
 716
 717
 718
 719
 720
 721
 722
 723
 724
 725
 726
 727
 728
 729
 730
 731
 732
 733
 734
 735
 736
 737
 738
 739
 740
 741
 742
 743
 744
 745
 746
 747
 748
 749
 750
 751
 752
 753
 754
 755
 756
 757
 758
 759
 760
 761
 762
 763
 764
 765
 766
 767
 768
 769
 770
 771
 772
 773
 774
 775
 776
 777
 778
 779
 780
 781
 782
 783
 784
 785
 786
 787
 788
 789
 790
 791
 792
 793
 794
 795
 796
 797
 798
 799
 800
 801
 802
 803
 804
 805
 806
 807
 808
 809
 810
 811
 812
 813
 814
 815
 816
 817
 818
 819
 820
 821
 822
 823
 824
 825
 826
 827
 828
 829
 830
 831
 832
 833
 834
 835
 836
 837
 838
 839
 840
 841
 842
 843
 844
 845
 846
 847
 848
 849
 850
 851
 852
 853
 854
 855
 856
 857
 858
 859
 860
 861
 862
 863
 864
 865
 866
 867
 868
 869
 870
 871
 872
 873
 874
 875
 876
 877
 878
 879
 880
 881
 882
 883
 884
 885
 886
 887
 888
 889
 890
 891
 892
 893
 894
 895
 896
 897
 898
 899
 900
 901
 902
 903
 904
 905
 906
 907
 908
 909
 910
 911
 912
 913
 914
 915
 916
 917
 918
 919
 920
 921
 922
 923
 924
 925
 926
 927
 928
 929
 930
 931
 932
 933
 934
 935
 936
 937
 938
 939
 940
 941
 942
 943
 944
 945
 946
 947
 948
 949
 950
 951
 952
 953
 954
 955
 956
 957
 958
 959
 960
 961
 962
 963
 964
 965
 966
 967
 968
 969
 970
 971
 972
 973
 974
 975
 976
 977
 978
 979
 980
 981
 982
 983
 984
 985
 986
 987
 988
 989
 990
 991
 992
 993
 994
 995
 996
 997
 998
 999
 1000
 1001
 1002
 1003
 1004
 1005
 1006
 1007
 1008
 1009
 1010
 1011
 1012
 1013
 1014
 1015
 1016
 1017
 1018
 1019
 1020
 1021
 1022
 1023
 1024
 1025
 1026
 1027
 1028
 1029
 1030
 1031
 1032
 1033
 1034
 1035
 1036
 1037
 1038
 1039
 1040
 1041
 1042
 1043
 1044
 1045
 1046
 1047
 1048
 1049
 1050
 1051
 1052
 1053
 1054
 1055
 1056
 1057
 1058
 1059
 1060
 1061
 1062
 1063
 1064
 1065
 1066
 1067
 1068
 1069
 1070
 1071
 1072
 1073
 1074
 1075
 1076
 1077
 1078
 1079
 1080
 1081
 1082
 1083
 1084
 1085
 1086
 1087
 1088
 1089
 1090
 1091
 1092
 1093
 1094
 1095
 1096
 1097
 1098
 1099
 1100
 1101
 1102
 1103
 1104
 1105
 1106
 1107
 1108
 1109
 1110
 1111
 1112
 1113
 1114
 1115
 1116
 1117
 1118
 1119
 1120
 1121
 1122
 1123
 1124
 1125
 1126
 1127
 1128
 1129
 1130
 1131
 1132
 1133
 1134
 1135
 1136
 1137
 1138
 1139
 1140
 1141
 114

(16)	特開2003-131772 30	<p>[0093] 本明細書で掲じた主な要素は、以下の通りである。</p> <p><prompt...> 音声合成構成およびプロンプトを再生成する場合</p> <p><reco...> 認識装置構成ならびに認識の実行および後処理をする場合</p> <p><grammar...> 入力文法リソースを指定する場合</p> <p>[0094] 2 Reco</p> <p>Reco要素は、可能なユーザ入力および入力結果を処理するための手段を指定する際に使用する。したがって、主な要素は<grammar>および<bind>であり、該装置のプロパティを構成するためのリソースを含む。</p> <p>[0095] Reco要素は、StartおよびStopメソッドを介してアップレベルプラグザで、あるいはモジュール3.2.4は、開発者によって指定されたプラグアットを用いて、クリアント側マークアップおよびスマートカード情報を取得するなどの高ディアイアログモジュールは、アップ例を用いて図6に図示したクリエイターカード情報を取得するなどの高ディアイアログモジュールは、アップ例をオーサリング中の開発者が使用するために、サーバ側例が記述装置3.2.4に格納する場合と同様に実施が可能である。一般に、高ディアイアログモジュール3.2.4は、開発者によって指定されたプラグアットを用いて、クリアント側マークアップおよびスマートカード情報を取得するための高ディアイアログモジュールのシナリオで動作的に生成する。高ディアイアログモジュールは、開発者のニーズに合うようなクリアント側マークアップができるためのバラメータを含むことができる。サーバ側プラグインモジュール3.2.0で使用するASP+ページの例が、図13に示されている。</p> <p>[0096] 2.1 Recoコン텐츠</p> <p>Reco要素には、1つまたは複数の文法および任意選択および認証プラットフォームの動作を正式するため、特定のモード、すなわちAutomatic、Manual、またはMultipleを採用する。また、または「multiple」を採用することも可能である。</p> <p>[0097] 2.1 Recoコン텐츠</p> <p>Reco要素には、1つまたは複数の文法および任意選択および認証のセットが含まれ、これが認証結果を検査し、直に関連する部分を含む側のページにコピーする。</p> <p>[0098] 2.1.1 <grammar></p> <p>以下のタグは、ドキュメントが音声を入力または出力供体として使用できるようにする、マークアップ要素のセットである。タグは、 XHTML、cHTML、SMIL、WML、などのような任意のSGML構造となる。XMLと同様に構成される、本明細書で使用されるタグは、ワンド社RecomendのMicrosoft Corporationから市販されているよく知られたメソッドである。SAPI 5.0と同様である。タグ、要素、イベント、属性、プロパティ、戻り値などは、単なる例示的なものであって、限定的なもののみではなくない。本明細書では音声およびDTMF認識について例示しているが、同様のタグを他の記載形態にも提供することができる。</p>
(17)	特開2003-131772 32	<p>[0099] 本明細書で掲じた主な要素は、以下の通りである。同じ名前を持つどんな規則も上書きされる。</p> <p>[0100] 属性：</p> <ul style="list-style-type: none"> • src : インライン文法が指定される場合、任意選択である。文法のURIが含まれる。文法内のすべての最高位規則は、認識文脈に関してデフォルト時に活動状態である。 • bind : 認識結果を指すための意味マークアップ言語(SML)を含むXMLドキュメントであつてよい。そのコントンツには、意味的、および機能的、および強度が含まれる。SMLは代替の認識選択肢をふくむことでもできる(N級複数結果の場合と同様)。「SMLドキュメントの例を以下に示す。 <p>[0101] lang ID</p> <p>lang ID : 任意選択である。どの音音声エンジンを使用するべきであるかを示す文字列。文字列オーナーは、xml:langを定義する。たとえば、lang ID='en-us'は米国英語を示す。この属性の値は米国英語である。</p> <p>[0102] grammar</p> <p>grammar : 例を以下に示す。</p> <pre> <grammar src="FrontCity.xml" /> or <grammar> <rule level="active"> <from /> <rule id="clites" /> </rule> <rule name="clites"> <from /> <rule id="l1" /> <rule id="l2" /> </rule> </grammar> <travel text="シトルからボストンに行きたい"> <origin_city confidence="45">シトル </origin_city> <dest_city confidence="35">ボストン </dest_city> </travel> </smil> </pre> <p>[0103] 2.1.2 <bind>要素</p> <p>Bind要素によって消費される認識結果は、SMLから文脈結果からその値をページに結合するのに使用される。</p> <p>[0104] bind</p> <p>Bind要素によって消費される認識結果は、SMLを指すための意味マークアップ言語(SML)を含むXMLドキュメントであつてよい。そのコントンツには、意味的、および機能的、および強度が含まれる。SMLは代替の認識選択肢をふくむことでもできる(N級複数結果の場合と同様)。「SMLドキュメントの例を以下に示す。</p>
(18)	特開2003-131772 33	<p>[0105] 文字での認識は、意味マークアップ言語またはSMLでXMLドキュメントを作成することが決定されているため、SMLドキュメントからbindされたalueコンテンツが割り当てられる要素(W3C SML 2.0の場合と同様)。</p> <p>[0106] 属性：</p> <ul style="list-style-type: none"> • targetElement : 必須。SMLからのvalueコンテンツが割り当てられる要素。 • targetAttribute : 任意選択。SMLからのtarget属性が割り当たるタグの属性。 <p>[0107] 値</p> <p>target要素は直近参照される場合。</p>

(18) 特開2003-131772
34
33
Name属性を使用する場合と同様)。指定されない場合は、**appendChild**方法であり、**bind**動作のデフォルトの値は*value*。
test：任意選択。認証結果が割り当てられる条件を示すXML Pattern (W3C XML DOM指定の場合と同様) 文字列。デフォルト条件は*true*。
value：必須。ターゲット要素に割り当てられる認証結果ドキュメントからの値を指定する、XPath (W3C XML DOM指定の場合と同様) 文字列。
[10107] 例：上記XMLの表りが与えられると、以下の**recognition**、**origin-city**とページトページ要素と**targetElement**に転送するために**bind**を使用する。
<input name="txtBoxOrigin" type="text"/>
<input name="txtBoxDest" type="text"/>

```
(recogId="travel")
  (grammar src="/city.xml" /)
  (bind targetElement="txtBoxOrigin"
        value="//origin_city" /)
  (bind targetElement="txtBoxDest"
        value="//dest_city" /)

```

[10108] この**bind**は、以下の例に示されるよう^{*}
 <bind targetElement="txtBoxDest"
 value="//dest_city"
 test="/xm/dstCity@confidence \$gt; 40" />

[10109] **bind**要素は、**download**または**babbleTimeOut**で認証結果を処理する単純な宣言後出後に結果を反映されなければならないミニ秒単位の時間。**automatic**および**singlemode**属性がある。より複雑な処理の場合、**appendChild**モードで**rec**オブジェクトラウザがサポートする**reco** DOMオブジェクトは、プログラムに基づくスクリプト分析および認証コードの後処理を可能にするために、**onReco**イベントは、**babbleTimeOut**モードでの**rec**オブジェクトがサポートされる。このタイムアウトは苦戦検出から停止コールまでの時間枠に適用される。[10110] 他の**rec**オブジェクトは、それぞれの結果が反映され、または苦戦検出から停止コールまでの時間枠に適用される。これと組み合った結果が超えると、エラーが発生したか否かに従って、様々なイベントが投げられる。認証装置への接続を実現する。

[10111] 2. 2. 1 属性
 以下の**Rec**の属性は、すべてのブラウザによってサポートされる。ただし他のなんらかの理由でタイムアウトを超えた場合は、**babbleTimeOut**モードを実現する。

[10111] 2. 2. 1 属性
 以下の**Rec**の属性は、ダイアログターン (**dialogTurn**) に関する苦戦装置を構成するのに使用される。

[10111] 2. 2. 2 属性
 以下の**Rec**の属性は、**babbleTimeOut**で認証結果を反映するまでのミニ秒出力の時間。この値は苦戦ブラウザがサポートする場合、苦戦ブラウザによってサポートされる。

* ただし他のなんらかの理由でタイムアウトを超えた場合は、**babbleTimeOut**属性が無効となる。

(19) 特開2003-131772
35
 ベントが投げられ、これは、分散環境でのネットワークまたは認証装置の履歴に応じるものである。**multiple**モードでの**rec**の場合は、**babblerimeout**属性を使用する場合と同様に、この時間枠は、それぞれの結果が反映されるかまたは他のイベント後に再開される。**bind**モードは、マークアップタグの並びおよびその処理を、**client**イベントに対するものであらぶ。文字符列は、「smx」メッセージオブジェクトから得られる。[10112] 例：此取り専用。認証プラットフォームから戻される状態コード。可能な値は、認証が成功した場合は0、または失敗値は-1から-4 (**Start**メソッド (セッション2、3、1) および**Activate**メソッド (セッション2、3、4) の可能な例外に定義されたどおり)、ならびに認証装置イベントの取り時に設定される状態値-1から-15 (2、4を参照) である。

[10113] 2. 3 オブジェクトメント

status：此取り専用。認証プラットフォームから戻される状態コード。可能な値は、認証が成功した場合は0、または失敗値は-1から-4 (**Start**メソッド (セッション2、3、1) および**Activate**メソッド (セッション2、3、4) の可能な例外に定義されたどおり)、ならびに認証装置イベントの取り時に設定される状態値-1から-15 (2、4を参照) である。

[10114] 2. 3. 1 **Start**

認証文脈に対してすべての文法最高位規則 (アップレベルブラケット) を活動化および非活動化することができます。たとえば次のようになります。

```
protocol://yourspeechplatform?targetEnergyThreshold=30
  id=0.5
  langID:任意選択。どの言語音声エンジンを使用するべきかを示す文字列。文字列フォーマットは、xml:lang属性は、当該言語を示す。この属性は、grammar要素には、非ゼロのstatusコードを設定し、確実にはonNoRecoイベントを開始 (fire) する。可能な語番には、文法がない (reco status=-1)、文法のコンバイル障害と同じく様々な理由で発生することのある文法のロード障害がある。
```

[10115] 例：

Object.**Start**()

戻り値：

[10116] 2. 3. 2 **Stop**

Stopメソッドは、認証プロセスを終了させるためのコードである。Recoオブジェクトは音声の信号を停止し、認証装置を受け取った音声に関する認識結果を、XML DOMノードオブジェクト内に保持される認証結果 (例も認証されない場合、プロパティはnull) を戻す。

[10117] 2. 4. 2 **Text**

Text：認証ノ替込み。認証された音楽のテキストを出すと、ブラウザによって**onTimeOut**を呼び出した後に検出終了時点で**reco**オブジェクト

37 を停止するので、`automaticMode`で典型的な認識のために明示的に使用される必要がないことに留意されたい。) `Reco`が開始されていない場合、コードは無効である。 [0117] 構文： `Object. Stop ()` 戻り値： なし 例外： なし [0118] 2. 3. 3 `Cancel ()` ; `Cancel` メソッドは、音声を認識装置に送るのを停止し、文法を非活動化して認識装置を解除し、どのような認識結果も無効である。 [0119] 構文： `Object. Cancel ()` 戻り値： なし [0120] 2. 3. 4 `Activate ()` ; `Activate` メソッドは、文脈自由文法 (CFG) の最高位規則を活動化する。活動化は、`start` ループプロセス中は無効となるため、認識が開始される前に呼び出さなければならぬ。明示的に非活動化されない認識文脈に関するすべての文法最高位規則は、すでに活動状態として処理されていることに留意されたい。 [0121] 構文： `Object. Activate (strName) ;` パラメータ： なし [0122] 2. 3. 5 `Deactivate ()` ; このメソッドは、文法の最高位規則を非活動化する。規則が存在しない場合、メソッドは無効である。 [0123] 構文： `Object. Deactivate (strName)` ; 戻り値： なし ※ strName : 必須。活動化される規則の名前 戻り値： なし	38 * * strName : 必須。活動化される規則の名前 戻り値： なし 例外： なし [0124] 2. 4. 1 `Reco` イベント DOMオブジェクトは以下のイベントをサポートするが、そのハンドラは、`reco`要素の属性として指定することができます。 [0125] 2. 4. 1 `onReco` : このイベントは、認識装置がブラウザに使用できる認識結果を得たときに発生する可能性のある問題を反映する。認識はデータに接続され、リースを消費する (2. 3. 2 を参照)。`onReco`は、典型的には、認識結果のプログラムに基づいた分析と、結果をページに入れる処理に使用される。 [0126] 構文： ※ [0127] [0128] イベントオブジェクト情報： ※ [0129] ※ [表 2]	39 </reco> <script>!(CDATA[function processCityRecognition () { smResult = event.srcElement.recognition; origNode = smResult.selectingNode("//origInCity"); if (origNode != null) txtBoxOrigin.value = origNode.text; destNode = smResult.selectingNode("//destCity"); destNode.text; if (destNode != null) txtBoxDest.value = destNode.text; }]></script> [0132] 2. 4. 2 `onSilence` : * * a t i c r e c o g n i t i o nモードの場合に認識がロセスを自動的に取り消す。 いんは、RecoのinitialTime属性で指定された持続時間 (2. 2. 1を参照) の [0133] 構文： 前に、認識プラットフォームによって検出された音声の [0134] ないイベントを処理する。このイベントは、autom * 20 [#3] インラインHTML. [0134] 2. 4. 3 `onTimeout` : なしこのイベントは、`onSilence="handle"`を Object.onSilence="handle" イベントローバイ (BOM)Scriptの属性 Object.onHandle="handle" デフォルト動作 [0135] イベントオブジェクト情報： ※ [表 4] [0136] 構文： ※ [0137] [0138] 2. 4. 3 `onTimeout` : なし ※ [表 5] ※ [表 6] [0139] このイベントは、認識処理を自動的に取り消す。 ★で発生する可能性のある問題を反映する。 認識が開始されたにもかかわらず、`onTimeout` (2. 2. 1を参照)によって指定された時間内に認識なしで処理が停止した場合、音声認識プラットフォームによって検出されたイベントも処理する (1.1)。 認識が完了する前に満了となった`maxLength`属性 (2. 2. 1を参照)に指定された時間内を超過する可能性があるため、タグインターフェリによって投げられたイベントを処理する。 典型的には、分散アーキテクチャ★ ※ [表 7] [0140] [0141] イベントオブジェクト情報： ※ [表 8] [0142] ※ [表 9] ※ [表 10] ※ [表 11] ※ [表 12] ※ [表 13] ※ [表 14] ※ [表 15] ※ [表 16] ※ [表 17] ※ [表 18] ※ [表 19] ※ [表 20] ※ [表 21] ※ [表 22] ※ [表 23] ※ [表 24] ※ [表 25] ※ [表 26] ※ [表 27] ※ [表 28] ※ [表 29] ※ [表 30] ※ [表 31] ※ [表 32] ※ [表 33] ※ [表 34] ※ [表 35] ※ [表 36] ※ [表 37] ※ [表 38] ※ [表 39] ※ [表 40] ※ [表 41] ※ [表 42] ※ [表 43] ※ [表 44] ※ [表 45] ※ [表 46] ※ [表 47] ※ [表 48] ※ [表 49] ※ [表 50] ※ [表 51] ※ [表 52] ※ [表 53] ※ [表 54] ※ [表 55] ※ [表 56] ※ [表 57] ※ [表 58] ※ [表 59] ※ [表 60] ※ [表 61] ※ [表 62] ※ [表 63] ※ [表 64] ※ [表 65] ※ [表 66] ※ [表 67] ※ [表 68] ※ [表 69] ※ [表 70] ※ [表 71] ※ [表 72] ※ [表 73] ※ [表 74] ※ [表 75] ※ [表 76] ※ [表 77] ※ [表 78] ※ [表 79] ※ [表 80] ※ [表 81] ※ [表 82] ※ [表 83] ※ [表 84] ※ [表 85] ※ [表 86] ※ [表 87] ※ [表 88] ※ [表 89] ※ [表 90] ※ [表 91] ※ [表 92] ※ [表 93] ※ [表 94] ※ [表 95] ※ [表 96] ※ [表 97] ※ [表 98] ※ [表 99] ※ [表 100] ※ [表 101] ※ [表 102] ※ [表 103] ※ [表 104] ※ [表 105] ※ [表 106] ※ [表 107] ※ [表 108] ※ [表 109] ※ [表 110] ※ [表 111] ※ [表 112] ※ [表 113] ※ [表 114] ※ [表 115] ※ [表 116] ※ [表 117] ※ [表 118] ※ [表 119] ※ [表 120] ※ [表 121] ※ [表 122] ※ [表 123] ※ [表 124] ※ [表 125] ※ [表 126] ※ [表 127] ※ [表 128] ※ [表 129] ※ [表 130] ※ [表 131] ※ [表 132] ※ [表 133] ※ [表 134] ※ [表 135] ※ [表 136] ※ [表 137] ※ [表 138] ※ [表 139] ※ [表 140] ※ [表 141] ※ [表 142] ※ [表 143] ※ [表 144] ※ [表 145] ※ [表 146] ※ [表 147] ※ [表 148] ※ [表 149] ※ [表 150] ※ [表 151] ※ [表 152] ※ [表 153] ※ [表 154] ※ [表 155] ※ [表 156] ※ [表 157] ※ [表 158] ※ [表 159] ※ [表 160] ※ [表 161] ※ [表 162] ※ [表 163] ※ [表 164] ※ [表 165] ※ [表 166] ※ [表 167] ※ [表 168] ※ [表 169] ※ [表 170] ※ [表 171] ※ [表 172] ※ [表 173] ※ [表 174] ※ [表 175] ※ [表 176] ※ [表 177] ※ [表 178] ※ [表 179] ※ [表 180] ※ [表 181] ※ [表 182] ※ [表 183] ※ [表 184] ※ [表 185] ※ [表 186] ※ [表 187] ※ [表 188] ※ [表 189] ※ [表 190] ※ [表 191] ※ [表 192] ※ [表 193] ※ [表 194] ※ [表 195] ※ [表 196] ※ [表 197] ※ [表 198] ※ [表 199] ※ [表 200] ※ [表 201] ※ [表 202] ※ [表 203] ※ [表 204] ※ [表 205] ※ [表 206] ※ [表 207] ※ [表 208] ※ [表 209] ※ [表 210] ※ [表 211] ※ [表 212] ※ [表 213] ※ [表 214] ※ [表 215] ※ [表 216] ※ [表 217] ※ [表 218] ※ [表 219] ※ [表 220] ※ [表 221] ※ [表 222] ※ [表 223] ※ [表 224] ※ [表 225] ※ [表 226] ※ [表 227] ※ [表 228] ※ [表 229] ※ [表 230] ※ [表 231] ※ [表 232] ※ [表 233] ※ [表 234] ※ [表 235] ※ [表 236] ※ [表 237] ※ [表 238] ※ [表 239] ※ [表 240] ※ [表 241] ※ [表 242] ※ [表 243] ※ [表 244] ※ [表 245] ※ [表 246] ※ [表 247] ※ [表 248] ※ [表 249] ※ [表 250] ※ [表 251] ※ [表 252] ※ [表 253] ※ [表 254] ※ [表 255] ※ [表 256] ※ [表 257] ※ [表 258] ※ [表 259] ※ [表 260] ※ [表 261] ※ [表 262] ※ [表 263] ※ [表 264] ※ [表 265] ※ [表 266] ※ [表 267] ※ [表 268] ※ [表 269] ※ [表 270] ※ [表 271] ※ [表 272] ※ [表 273] ※ [表 274] ※ [表 275] ※ [表 276] ※ [表 277] ※ [表 278] ※ [表 279] ※ [表 280] ※ [表 281] ※ [表 282] ※ [表 283] ※ [表 284] ※ [表 285] ※ [表 286] ※ [表 287] ※ [表 288] ※ [表 289] ※ [表 290] ※ [表 291] ※ [表 292] ※ [表 293] ※ [表 294] ※ [表 295] ※ [表 296] ※ [表 297] ※ [表 298] ※ [表 299] ※ [表 300] ※ [表 301] ※ [表 302] ※ [表 303] ※ [表 304] ※ [表 305] ※ [表 306] ※ [表 307] ※ [表 308] ※ [表 309] ※ [表 310] ※ [表 311] ※ [表 312] ※ [表 313] ※ [表 314] ※ [表 315] ※ [表 316] ※ [表 317] ※ [表 318] ※ [表 319] ※ [表 320] ※ [表 321] ※ [表 322] ※ [表 323] ※ [表 324] ※ [表 325] ※ [表 326] ※ [表 327] ※ [表 328] ※ [表 329] ※ [表 330] ※ [表 331] ※ [表 332] ※ [表 333] ※ [表 334] ※ [表 335] ※ [表 336] ※ [表 337] ※ [表 338] ※ [表 339] ※ [表 340] ※ [表 341] ※ [表 342] ※ [表 343] ※ [表 344] ※ [表 345] ※ [表 346] ※ [表 347] ※ [表 348] ※ [表 349] ※ [表 350] ※ [表 351] ※ [表 352] ※ [表 353] ※ [表 354] ※ [表 355] ※ [表 356] ※ [表 357] ※ [表 358] ※ [表 359] ※ [表 360] ※ [表 361] ※ [表 362] ※ [表 363] ※ [表 364] ※ [表 365] ※ [表 366] ※ [表 367] ※ [表 368] ※ [表 369] ※ [表 370] ※ [表 371] ※ [表 372] ※ [表 373] ※ [表 374] ※ [表 375] ※ [表 376] ※ [表 377] ※ [表 378] ※ [表 379] ※ [表 380] ※ [表 381] ※ [表 382] ※ [表 383] ※ [表 384] ※ [表 385] ※ [表 386] ※ [表 387] ※ [表 388] ※ [表 389] ※ [表 390] ※ [表 391] ※ [表 392] ※ [表 393] ※ [表 394] ※ [表 395] ※ [表 396] ※ [表 397] ※ [表 398] ※ [表 399] ※ [表 400] ※ [表 401] ※ [表 402] ※ [表 403] ※ [表 404] ※ [表 405] ※ [表 406] ※ [表 407] ※ [表 408] ※ [表 409] ※ [表 410] ※ [表 411] ※ [表 412] ※ [表 413] ※ [表 414] ※ [表 415] ※ [表 416] ※ [表 417] ※ [表 418] ※ [表 419] ※ [表 420] ※ [表 421] ※ [表 422] ※ [表 423] ※ [表 424] ※ [表 425] ※ [表 426] ※ [表 427] ※ [表 428] ※ [表 429] ※ [表 430] ※ [表 431] ※ [表 432] ※ [表 433] ※ [表 434] ※ [表 435] ※ [表 436] ※ [表 437] ※ [表 438] ※ [表 439] ※ [表 440] ※ [表 441] ※ [表 442] ※ [表 443] ※ [表 444] ※ [表 445] ※ [表 446] ※ [表 447] ※ [表 448] ※ [表 449] ※ [表 450] ※ [表 451] ※ [表 452] ※ [表 453] ※ [表 454] ※ [表 455] ※ [表 456] ※ [表 457] ※ [表 458] ※ [表 459] ※ [表 460] ※ [表 461] ※ [表 462] ※ [表 463] ※ [表 464] ※ [表 465] ※ [表 466] ※ [表 467] ※ [表 468] ※ [表 469] ※ [表 470] ※ [表 471] ※ [表 472] ※ [表 473] ※ [表 474] ※ [表 475] ※ [表 476] ※ [表 477] ※ [表 478] ※ [表 479] ※ [表 480] ※ [表 481] ※ [表 482] ※ [表 483] ※ [表 484] ※ [表 485] ※ [表 486] ※ [表 487] ※ [表 488] ※ [表 489] ※ [表 490] ※ [表 491] ※ [表 492] ※ [表 493] ※ [表 494] ※ [表 495] ※ [表 496] ※ [表 497] ※ [表 498] ※ [表 499] ※ [表 500] ※ [表 501] ※ [表 502] ※ [表 503] ※ [表 504] ※ [表 505] ※ [表 506] ※ [表 507] ※ [表 508] ※ [表 509] ※ [表 510] ※ [表 511] ※ [表 512] ※ [表 513] ※ [表 514] ※ [表 515] ※ [表 516] ※ [表 517] ※ [表 518] ※ [表 519] ※ [表 520] ※ [表 521] ※ [表 522] ※ [表 523] ※ [表 524] ※ [表 525] ※ [表 526] ※ [表 527] ※ [表 528] ※ [表 529] ※ [表 530] ※ [表 531] ※ [表 532] ※ [表 533] ※ [表 534] ※ [表 535] ※ [表 536] ※ [表 537] ※ [表 538] ※ [表 539] ※ [表 540] ※ [表 541] ※ [表 542] ※ [表 543] ※ [表 544] ※ [表 545] ※ [表 546] ※ [表 547] ※ [表 548] ※ [表 549] ※ [表 550] ※ [表 551] ※ [表 552] ※ [表 553] ※ [表 554] ※ [表 555] ※ [表 556] ※ [表 557] ※ [表 558] ※ [表 559] ※ [表 560] ※ [表 561] ※ [表 562] ※ [表 563] ※ [表 564] ※ [表 565] ※ [表 566] ※ [表 567] ※ [表 568] ※ [表 569] ※ [表 570] ※ [表 571] ※ [表 572] ※ [表 573] ※ [表 574] ※ [表 575] ※ [表 576] ※ [表 577] ※ [表 578] ※ [表 579] ※ [表 580] ※ [表 581] ※ [表 582] ※ [表 583] ※ [表 584] ※ [表 585] ※ [表 586] ※ [表 587] ※ [表 588] ※ [表 589] ※ [表 590] ※ [表 591] ※ [表 592] ※ [表 593] ※ [表 594] ※ [表 595] ※ [表 596] ※ [表 597] ※ [表 598] ※ [表 599] ※ [表 600] ※ [表 601] ※ [表 602] ※ [表 603] ※ [表 604] ※ [表 605] ※ [表 606] ※ [表 607] ※ [表 608] ※ [表 609] ※ [表 610] ※ [表 611] ※ [表 612] ※ [表 613] ※ [表 614] ※ [表 615] ※ [表 616] ※ [表 617] ※ [表 618] ※ [表 619] ※ [表 620] ※ [表 621] ※ [表 622] ※ [表 623] ※ [表 624] ※ [表 625] ※ [表 626] ※ [表 627] ※ [表 628] ※ [表 629] ※ [表 630] ※ [表 631] ※ [表 632] ※ [表 633] ※ [表 634] ※ [表 635] ※ [表 636] ※ [表 637] ※ [表 638] ※ [表 639] ※ [表 640] ※ [表 641] ※ [表 642] ※ [表 643] ※ [表 644] ※ [表 645] ※ [表 646] ※ [表 647] ※ [表 648] ※ [表 649] ※ [表 650] ※ [表 651] ※ [表 652] ※ [表 653] ※ [表 654] ※ [表 655] ※ [表 656] ※ [表 657] ※ [表 658] ※ [表 659] ※ [表 660] ※ [表 661] ※ [表 662] ※ [表 663] ※ [表 664] ※ [表 665] ※ [表 666] ※ [表 667] ※ [表 668] ※ [表 669] ※ [表 670] ※ [表 671] ※ [表 672] ※ [表 673] ※ [表 674] ※ [表 675] ※ [表 676] ※ [表 677] ※ [表 678] ※ [表 679] ※ [表 680] ※ [表 681] ※ [表 682] ※ [表 683] ※ [表 684] ※ [表 685] ※ [表 686] ※ [表 687] ※ [表 688] ※ [表 689] ※ [表 690] ※ [表 691] ※ [表 692] ※ [表 693] ※ [表 694] ※ [表 695] ※ [表 696] ※ [表 697] ※ [表 698] ※ [表 699] ※ [表 700] ※ [表 701] ※ [表 702] ※ [表 703] ※ [表 704] ※ [表 705] ※ [表 706] ※ [表 707] ※ [表 708] ※ [表 709] ※ [表 710] ※ [表 711] ※ [表 712] ※ [表 713] ※ [表 714] ※ [表 715] ※ [表 716] ※ [表 717] ※ [表 718] ※ [表 719] ※ [表 720] ※ [表 721] ※ [表 722] ※ [表 723] ※ [表 724] ※ [表 725] ※ [表 726] ※ [表 727] ※ [表 728] ※ [表 729] ※ [表 730] ※ [表 731] ※ [表 732] ※ [表 733] ※ [表 734] ※ [表 735] ※ [表 736] ※ [表 737] ※ [表 738] ※ [表 739] ※ [表 740] ※ [表 741] ※ [表 742] ※ [表 743] ※ [表 744] ※ [表 745] ※ [表 746] ※ [表 747] ※ [表 748] ※ [表 749] ※ [表 750] ※ [表 751] ※ [表 752] ※ [表 753] ※ [表 754] ※ [表 755] ※ [表 756] ※ [表 757] ※ [表 758] ※ [表 759] ※ [表 760] ※ [表 761] ※ [表 762] ※ [表 763] ※ [表 764] ※ [表 765] ※ [表 766] ※ [表 767] ※ [表 768] ※ [表 769] ※ [表 770] ※ [表 771] ※ [表 772] ※ [表 773] ※ [表 774] ※ [表 775] ※ [表 776] ※ [表 777] ※ [表 778] ※ [表 779] ※ [表 780] ※ [表 781] ※ [表 782] ※ [表 783] ※ [表 784] ※ [表 785] ※ [表 786] ※ [表 787] ※ [表 788] ※ [表 789] ※ [表 790] ※ [表 791] ※ [表 792] ※ [表 793] ※ [表 794] ※ [表 795] ※ [表 796] ※ [表 797] ※ [表 798] ※ [表 799] ※ [表 800] ※ [表 801] ※ [表 802] ※ [表 803] ※ [表 804] ※ [表 805] ※ [表 806] ※ [表 807] ※ [表 808] ※ [表 809] ※ [表 810] ※ [表 811] ※ [表 812] ※ [表 813] ※ [表 814] ※ [表 815] ※ [表 816] ※ [表 817] ※ [表 818] ※ [表 819] ※ [表 820] ※ [表 821] ※ [表 822] ※ [表 823] ※ [表 824]

(22)	特開2003-131772	<p>41</p> <table border="1"> <tr> <td>ハンドル</td><td>なし</td></tr> <tr> <td>呼び出し方法</td><td>回路構成する前に machine 属性によって回路された回路を切替えた場合は、プラグによって接続する。</td></tr> <tr> <td>デスクホール動作</td><td>two status & 12に接続</td></tr> </table> <p>[0143] イベントプロバイディ：イベントハンドラはプロバイディを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会する必要のある場合はある。特定の値を確認するために、たとえば HTML書式制御の値をデリフェンスする必要がある。その場合は、prompt が送出される前に、「txttBoxOrigin」要素および「txttBoxDest」要素の「value」属性がテキストに挿入される。</p> <p>[0145] Value 属性 value:任意選択。ドキュメント内にある要素の直行先は、 value 属性を使用することができます。</p> <p>[0146] 属性： ・targetElement:任意選択。href または targetElement のいずれかを指定する。 ・value:任意選択。ドキュメント内にある要素の id。 ・targetAttribute:任意選択。値が取り出される要素の属性。 ・href:任意選択。音声認識プラットフォームによって投じられたは targetElement の両方で存在する場合、href が targetElement に優先される。</p> <p>[0147] targetElement の属性は、含む側のドキュメント内にある要素の参照を使用される。id が targetElement によって指定されている要素のコンテンツが要素の属性に表示されている。</p> <p>※ [表8]</p> <table border="1"> <tr> <td>ハンドル</td> <td>なし</td> </tr> <tr> <td>呼び出し方法</td> <td>監視装置が音声を検出したが、発話と解釈することができない。</td> </tr> <tr> <td>デフォルト動作</td> <td>nameプロパティを設定し、null 認識結果を戻す。statusコードは以下の通りに設定される。 status-13：音声が検出されたが音声が解釈できなか status-14：音声の音声が検出されなかった。(しきい値設定につ ては2.2.1のreject 属性を参照)。 status-15：音声が検出され解釈されたが、音声の検出 が他の不十分で解釈された。(しきい値設定につ ては2.2.1のreject 属性を参照)。</td> </tr> </table>	ハンドル	なし	呼び出し方法	回路構成する前に machine 属性によって回路された回路を切替えた場合は、プラグによって接続する。	デスクホール動作	two status & 12に接続	ハンドル	なし	呼び出し方法	監視装置が音声を検出したが、発話と解釈することができない。	デフォルト動作	nameプロパティを設定し、null 認識結果を戻す。statusコードは以下の通りに設定される。 status-13：音声が検出されたが音声が解釈できなか status-14：音声の音声が検出されなかった。(しきい値設定につ ては2.2.1のreject 属性を参照)。 status-15：音声が検出され解釈されたが、音声の検出 が他の不十分で解釈された。(しきい値設定につ ては2.2.1のreject 属性を参照)。
ハンドル	なし													
呼び出し方法	回路構成する前に machine 属性によって回路された回路を切替えた場合は、プラグによって接続する。													
デスクホール動作	two status & 12に接続													
ハンドル	なし													
呼び出し方法	監視装置が音声を検出したが、発話と解釈することができない。													
デフォルト動作	nameプロパティを設定し、null 認識結果を戻す。statusコードは以下の通りに設定される。 status-13：音声が検出されたが音声が解釈できなか status-14：音声の音声が検出されなかった。(しきい値設定につ ては2.2.1のreject 属性を参照)。 status-15：音声が検出され解釈されたが、音声の検出 が他の不十分で解釈された。(しきい値設定につ ては2.2.1のreject 属性を参照)。													
(23)	特開2003-131772	<p>42</p> <p>43</p> <p>44</p> <p>*場合、targetAttribute属性を使用して、targetElementに必要な属性を指定することができる。これまではHTML書式制御の値をデリフェンスするために役立つ。以下の例示では、prompt が送出される前に、「txttBoxOrigin」要素および「txttBoxDest」要素の「value」属性がテキストに挿入される。</p> <p>(prompt 1#<input type="text" value="txtBarBest"/>)</p> <p>行先は value targetElement="txtBarOrigin" value="value" /></p> <p>targetAttribute="value" /></p> <p>prompt</p> <p>から (value targetElement="txtBarBest" value="value" />)</p> <p>targetAttribute="value" /></p> <p>targetAttribute="value" /></p> <p>[0158] 3. 1. 3 音声ファイル value要素は、合成されたpromptの代わりに、あるいはその中で再生するために、事前に録音された音声ファイルを参照するのに使用することもできる。</p> <p>[0159] 3. 1. 4 参照読み込みプロンプト targetElementの両方で同時に合成および、ミリ秒単位の時間軸。デフォルトの値は無限であり、すなわち 30 秒の要件を備えたsrc属性を指定することで、以下のようになります。targetElementの両方で同時に合成および、URLを介して外部コンテンツを参照することができます。これは、ブラウザがサポートしているどんな種類の割り込みにも適用される。recoが開始された時点、キー操作またはエネルギーベースのbargeIn時間のどちらが使用可能であるかに応じて、いずれかがこの方法で構成されます。</p> <p>・preetch:任意選択。ページがロードされたときに、promptをブラウザ側で同時に合成およびコンテンツをインラインで指定するだけで、以下のようになります。targetElementの両方で同時に合成および、URLを介して外部コンテンツを参照することができます。これは、ブラウザがサポートしているどの種類の割り込みにも適用される。recoが開始された時点、キー操作またはエネルギーベースのbargeIn時間のどちらが使用可能であるかに応じて、いずれかがこの方法で構成されます。</p> <p>・prompt id="Welcome" src="/ACMEetherPrompts#Welcome" /></p> <p>src属性のターゲットは、INLINEプロンプト用に指定された上記コンテンツのいずれかまたはすべてを保持することができます。</p> <p>[0160] 3. 2 属性およびプロパティ prompt要素は、以下の属性 (ダウンレベルブラウザ) およびプロパティ (ダウンレベルおよびアップレベルブラウザ) を保持する。</p> <p>[0162] 3. 2. 2 プロパイア アップレベルブラウザは、promptのDOMオブジェクト内で以下のプロパティをサポートする。</p> <p>・status: 録音専用。音声プラットフォームによって戻されるstatus属性を戻す。</p> <p>・bookmark: 録取り専用。発生した最新の合成音声ブックマークのテキストを保管する文字列オブジェクト。</p> <p>・bargain: 照会済みプロンプトのURl (3. 1. 4 を参照)。</p> <p>・bargain: 任意選択。整数。プロンプトの開始位置。</p> <p>・bargain: 任意選択。整数。システム出力によって戻されるstatusコード。</p> <p>・innerext: 録取り専用。音声プラットフォームに</p>												

(24)	特開2003-131772 46	<p>は、promptのテキスト表現形式(transcription)を提供し、これが合成装置に送られる。たとえば、promptが可聴版ファイルの再生を含む場合、このプロパティはそのpromptのテキストメッセージ(可聴版ファイルと共に格納されている場合)で多いを提供し、次いだときは、promptのデバイストーベーションを構成要素またはクリアントデバイス上で実行中のアプリケーションに提供することによって、表示されるか、そうでなければ使用される。innerTextプロパティは、動的コンテンツを含むpromptのテキストバージョンを提供する際にも使用できる。</p> <p>[0168] 3. 3. 3. Resume 例外：音声バッファをフラッシングせずに再生を再開する。再生が一時停止されていない場合、このメッセージは無効である。</p> <p>[0169] 構文： Object.Resume(); 戻り値： 例外：再開が失敗すると例外が投げられる。 [0170] 3. 3. 4. Stop 例外：音声が失敗するときに使用する停止。音声バッファが停止されなければ再生を停止し、音声バッファをフラッシングする。再生がすでに停止されている場合は音声バッファをフラッシングするだけである。</p> <p>[0171] 構文： Object.Stop(); 戻り値： 音声の音量および音質の変更(change)を実行することができる。</p> <p>[0172] 3. 3. 5. Change 例外：なし 再生の速度を上げ／または音量を変更する。Changeは再生中に呼び出すことができる。 [0173] 構文： Object.Change(speed, volume); パラメータ： speed：必須。変更する因数。speed = 2.0は現在の速度を2倍にする。volume = 0は現在の速度を半分にする。speed = 0.5はvolume = 0はデフォルト値を復元するという意味である。</p> <p>[0174] 構文： Object.Start([startText]); パラメータ： startText：合成装置に送信されるテキスト。存在する場合は、この引数がオブジェクトのコンテンツに優先される。</p> <p>[0175] 構文： 例外：音声バッファを停止または再生を一時停止する。このメッセージはサーバによってすでに解放されている場合はonCloseイベントが開始される。</p> <p>[0176] 3. 4. Promptイベント 例外：status = -1に設定すると、音声バッファがサーバによってすでに解放されている場合はonCloseオルト値を復元するという意味である。</p> <p>[0177] 3. 4. 1. onBookmark 例外： インラインHTML イベントプロパティ Object.onBookmark = handler Object.onBookmark = GetRef("handler"); 戻り値： ※40 [表9] ※40 [表9]</p>
(25)	特開2003-131772 48	<p>[0178] 構文： [0179]</p> <p>[0180] イベントオブジェクト情報 [0181]</p> <p>例外： 例外：表示された文字が闪烁する。 例外：闪烁マークが表示する。</p>
(26)	特開2003-131772 49	<p>[0182] 構文： [0183]</p>
(27)	特開2003-131772 50	<p>[0184] 構文： [0185]</p>

(26) 特開2003-131772

49 [0182] イベントプロパティ：イベントハンドラは * ば、エネルギー検出またはキーワード認識を決定するプロパティを直接受け取らないが、ハンドラはデータにとはプラットフォームの責任であることに留意された。このイベントハンドラを指定しても、`bargain` が選択するイベントオブジェクトを照会することができる。

[0183] 3. 2. onBargain : ユーザーが `Bargain` イベントが検出されると開始される。構文：
`[0184]`
`[0185] (bargain, bargain.onBargain="handler")`

50 [0186] イベントオブジェクト情報 : ※ [表12]

イベントプロパティ	<code>Object.onBargain = "handler"</code> <code>Object.onBargain = "GetRefHandler"</code>
-----------	--

51 [0187] イベントプロパティ：イベントハンドラは * 設定）に遭遇すると、開始される。
`[0188]`
`[0189] プロパティを直接受け取らないが、ハンドラはデータに`
`[0190] 開始するイベントオブジェクトを照会する。`
`[0191]`

52 [0192] イベントオブジェクト情報 : ★ [表13]

イベントプロパティ	<code>Object.onComplete = "handler"</code> <code>Object.onComplete = "GetRefHandler"</code>
-----------	--

53 [0193] イベントプロパティ：イベントハンドラは * ば、エネルギー検出またはキーワード認識を決定するプロパティを直接受け取らないが、ハンドラはデータにとはプラットフォームの責任であることに留意された。このイベントハンドラを指定しても、`bargain` が選択するイベントオブジェクトを照会することができます。

[0194] 3. 4. 3. onComplete : pr
`[0195]`
`[0196]`

54 [0197] イベントオブジェクト情報 : ★ [表14]

イベントプロパティ	<code>Object.onComplete = "handler"</code> <code>Object.onComplete = "GetRefHandler"</code>
-----------	--

55 [0198] txtBoxDest, value = event.srcElement.text;
`[0199]`
`[0200]`

56 [0201] `</script>`
`<body>`
`<input name="txtBoxOrigin" value="シアトル" type="text" />`

57 [0202] `<input name="txtBoxDest" type="text" />`

58 [0203] `<input name="confirm" onBargain="interrupt0" type="button" value="確認" />`

59 [0204] `<prompt id="confirm" onBargain="interrupt0" type="text" />`

60 [0205] `<From> <bookmark mark="mark_origin_city" />`

61 [0206] `<Value targetElement="origin" targetAttribute="value" />`

62 [0207] `<Bookmark mark="mark_dest_city" />` が行く先であることを復習してください。

63 [0208] `</prompt>`

64 [0209] `<reco onReco="ProcessCityConfirm0" />`

65 [0210] `<grammar src="/src/1033/cities.xml" />`

66 [0211] `</reco>`

67 [0212] `</body>`

68 [0213] `</html>`

69 [0214] 4. DTMF
`[0215]`
`[0216] DTMF認識オブジェクトを作成する。このオブジェクトはオンラインマークアップ言語構文を使用するかスクローリングでインスタンス化することができます。DTMFは活動化されたときに、promptオブジェクトにbargainイベントを開始させることができ`
`[0217] る。DTMF認識に關して以下で論じるタグおよびイベント。DTMF認識結果が割り当てられる要素（たとえばW3C SMIL 2.0の場合と同様）。`
`[0218] 2. 0の場合`
`[0219] 2. 0の場合`
`[0220] の場合に関係するものである。`

70 [0221] 4. 1. コンテンツ
`[0222] -dimgrammar : インライン式文である。`

71 [0223] 例：キーをテキストにマッピングする
`[0224] <input type="text" name="city"/>`

72 [0225] `<dtmf id="city_choice" timeout="2000" numDigits="1" />`

73 [0226] `<key value="1" /> シアトル`

74 [0227] `<key value="2" /> ポストン`

75 [0228] `</dtmfgrammar>`

76 [0229] `<bind targetElement="city" targetAttribute="value" />`

77 [0230] `</DIME>`

78 [0231] `[city_choice] が活動化されたときに、ユーザが1を押すと「Seattle」が、2を押すと「Boston」が入力ファイルに割り当てられ、それ以外の場合には何も割り当られない。`

79 [0232] `[0198]`

80 [0233] `function interrupt() {`

81 [0234] `mark = event.srcElement.bookmark;`

82 [0235] `}`

83 [0236] `function ProcessCityConfirm0() {`

84 [0237] `confirm.stop(); // オーディオバッファをフリッシュする。`

85 [0238] `if (mark == "mark_origin_city") {`

86 [0239] `txtBoxOrigin.value =`

87 [0240] `event.srcElement.text;`

特開2003-131772

例 2：複数フィールドの場合は DTMF の使用方法

53

```

<Input type="text" name="area_code" />
<Input type="text" name="phone_number" />
<DTMF id="areacode" numDigits="3"
      >Area Code:</DTMF>
<DTMF id="extension" activate()>
      Extension:</DTMF>
</DTMF>
<DTMF id="extention" numDigits="7">
      Extension:</DTMF>
</DTMF>

```

[0] 99

の例は、ユーザーが複数のファイルに入力する方法を *
例3：音声入力とDTMF入力の両方を使用可能にし、ユーザーがDTMFを聞
いた場合は音声入力を使用不能にする方法

```
<input type="text" name="credit_card_number" />
<prompt onBookmarkId="dtmf_start0" speech="Start 0"
        bareIn="0" />
<bookmark name="starting" />
</prompt>
<br>を貯留するか、クレジットカード番号を入力してください。
```

</prompt>
<DTMF id="dtmf" escape="#" length="16"
 interdigitTimeout="2000"

```
oneyoure= speech.Stop() >
<blind targetElement="credit_card_number" />
</blind>
<echo id="speech" >
<grammar src="/gma/1033/digits.xml" />
<child targetElement="credit card number" />
```

1.0.2000] 4. 2 属性およびプロパティ	<p>・ 2. 1 属性</p> <p>dtmF grammar : 必須。 DTMF 文法の URL</p> <p>・ 1.0.201] 4. 2. 2 プロパティ</p> <p>D T M F g r a m m a r : 読取り／書き込み</p> <p>・ 1.0.201] 4. 2. 3 ノードオブジェクト</p> <p>D T M F T M F を扱う XML DOM N o d e オブジェクト</p> <p>・ 1.0.201] 4. 2. 4 ノードオブジェクト</p> <p>XML 文字列への変換マトリックス (DTMF 文法とともに呼ぶ) が格納される。 フォルトの文法は以下のとおり。</p> <p>(grammar)</p>
30	<p>トの値は、先行タイプを実行可能にするための <code>!als</code></p> <p><code>e.</code></p> <p><code>escape</code></p> <p>読み取り／書き込み。 DTMF 読取りセッションを終了するための <code>escape</code>。 <code>escape</code> は 1 つのキーである。</p> <p><code>numDigits</code></p> <p>読み取り／書き込み。 DTMF 読取りセッションを終了するためのストロークの数。<code>escape</code> と <code>length</code> の両方が指定された場合は、どちらかの条件が満たされると DTMF セッションは終了する。</p>
</read>	
40	<p><code>(key value="0")\key)</code></p>

入力された場合は結果的に escape が含まれる

(key value="") (key)
/diagrammer)
· flush
· t e x t
空白で分離されたトークン文字列を構築している限り
専用文字列であって、各トークンはDTMF文法に従
て変換される。

(29)

の例は、ユーザが複数のファイルに入力する方法を^{*}
例3：音声入力とDTMF入力の両方を使用可能にし、ユーザがDTMFを聞
かせてもそれを選択不能にする方法

```
<input type="text" name="credit_card_number" />
<preempt onBookmark=difm,Start(): speech.Start() " >
    bargain=0 >
        <bookmark name="starting" />
        を覚悟するか、クレジットカード番号を入力してください。
    </preempt>
        <DIFM id="difm" escape="#" length="16" >
            interdigitimeout=2000" >
```

イニシエーションHTML	<DTMF ondtmfinput=“andler”>
イベントプロセス	Object.onkeypress = handler Object.onkeydown = C-KeyPress-onhandler

オブジェクト情報:		※【表16】
オブジェクト名	なし	
呼出名	タグチーン監査キーパーに接続する	
オブジェクト属性	押すたまごセグメント	

[0213] イベントプロパティ：イベントハンドラはプロパティを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができる。

[0214] 4. 2 onRecd

[0215] 構文：
[0216].

★ントは現在のDTMFオブジェクトにする。

インライン HTML	<DIV style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto; text-align: center;">終了すると開始される。このイベ☆ [段落]
イベントプロパティ	<pre>Object.onfocus = handleFocus Object.onblur = handleBlur</pre>

オブジェクト情報:		☆ [第18]
バイブル	なし	
叫び方/方法	ユーヨーが かわいい キーを押すか、またはキーストローク	
アフレール/顔作	押されたキーを押す	押されたキーを押す

【0219】イベントプロパティ：イベントハンドラはプロパティを直接操作しないが、ハンドラはデータに影響するイベントオブジェクトを監視することができます。
【0220】4.4.3 onTimeOut タイムアウト以前に完了イベントが受け取られないときにイベントは認証プロセスを自動的に停止される。このイベントは認証プロセスを開始する。

(30) 特開2003-131772
57 * [0222]
[0221] 構文：
* 【袋19】

オンラインHTML	<DTMFonTimeout>
イベントプロトコル	Object.onTimeout = handle
(EOMS-Sipの場合は	Object.onTimeout = GetOffHoldHandle);

[0223] イベントオブジェクト情報：
※【袋20】

[0224] 構文：
* 【袋21】

イベント動作	なし
呼び出し方法	指定されたタイムアウト内に、例のDTMFキーストロークも検出されたか、
デフォルト動作	なし

[0225] イベントプロトコル：イベントハンドラーはプロトコルを直接受け取らないが、ハンドラはデータに関するイベントオブジェクトを照会することができます。

[0226] 5. Call controlオブジェクト
例外：呼転送が失敗すると例外が投げられる。たとえば、発信担当者が話中の場合、こうした番号のファックト電話声プロトocolの電話インターフェース（呼、端末、および接続）を表す。このオブジェクトは、GUIプログラマでのwindowオブジェクトと同じように画面で表示される。したがって、電話オブジェクトの存続期間は常に限りがある。したがって、電話オブジェクトの存続期間は常にそれ自身と同じである。電話用の音声データは、呼び出し元が直接呼び出させるのはアプリケーションの役割である。基礎となる電話プラットフォームは、戻つてくみ再起動または配信は行わない。

[0227] 5. 1 プロトコル
例外：XML DOMノードオブジェクト。実態によって異なる。発呼者のアドレスである。PSTNの場合は、ANIとALIの組合せであってよい。VINの場合は、これが発呼者のIPアドレスである。ringBeforeAnswer属性呼に応答する前の呼び出し音の回数。デフォルト値は無限であり、これは、開発者が呼び出し元のために、具体的には以下のanswer（）メソッドを使用しなければならないことを意味する。電話局が着信呼を持ち行列に入れるのにADCを使用している場合、この番号は0に設定することができます。

[0228] 5. 2 メソッド
注：ここに示すメソッドはすべて同時に発生するものである。

[0229] 5. 2. 1 Transfer
例外：
注：転送する。ブラインド伝送の場合は、転送が完了するまで、システムはオリジナルの呼およびフリーステムソースを持続することができる。

[0230] 5. 2. 1 Transfer
例外：転換がない場合には例外が投げられる。この場合、answer（）メソッドは開始されない。

[0231] 5. 2. 5 Connect
例外：
注：第1当事者の外に向かう呼を開始する。

[0232] 5. 3. 1 onIncoming()
例外：呼び出し元が電話局が着信呼を受け取ったときに呼び出される。

[0233] 5. 3. 3 onHangup()
例外：呼び出し元が電話局が着信呼を受け取ったときに呼び出される。このイベントは、プログラムがHangupされたときには電話を切ったときに呼び出される。

[0234] 5. 4 Transfer()
例外：呼び出し元が電話セッションを解除するために所制御音声プロトocolが着信呼に応答したときに呼び出される。

[0235] 5. 5. 1 onAnswer()
例外：呼び出し元が電話用の音声番号に応答したときに呼び出される。

[0236] 5. 6 Record()
例外：ユーザの音声をファイルに記録する。

[0237] 5. 7 Silence()
例外：記録を停止するまでのミリ秒単位の時間。

[0238] 5. 8 Timeout()
例外：無音が検出された後に記録するための秒数。

[0239] 5. 9 maxTimeout()
例外：記録するための秒数。

[0240] 5. 10 maxTime()
例外：記録するための最大時間。

[0241] 5. 11 transfer()
例外：記録された結果のurlを返す。

[0242] 5. 12 endSilence()
例外：無音が検出された後に記録を停止するまでのミリ秒単位の時間。

[0243] 5. 13 start()
例外：記録するための秒数。

[0244] 5. 14 submit()
例外：記録された結果のurlを返す。

[0245] 5. 15 start()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0246] 5. 16 timeout()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0247] 5. 17 cancel()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0248] 5. 18 hangup()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0249] 5. 19 answer()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0250] 5. 20 silence()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0251] 5. 21 transfer()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0252] 5. 22 record()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0253] 5. 23 start()
例外：記録がuriに書き込まれないときに、例外が投げられる。

[0254] 5. 24 hangup()
例外：記録がuriに書き込まれないときに、例外が投げられる。

```

特例 2 0 0 3 - 1 3 1 7 2
62
63
64

61   function logIn_reco0 {
62     res = event.getSource();
63     pnode = res.selectSingleNode("//uid");
64     if (pnode != null) {
65       logon.user.value = pnode.xml;
66       pnode = res.selectSingleNode("//password");
67       if (pnode != null) {
68         logon.pass.value = pnode.xml;
69       }
70     }
71     function dtmf_reco0 {
72       res = event.getSource();
73       if (res == "user") {
74         logon.phn.value = res;
75       } else {
76         logon.phn.value = res;
77       }
78     }
79     </SCRIPT>
80     <SCRIPT for="callControl" event="onIncoming">
81     <!--
82       p_main.Start(); & login.Start(); dtmf.Start0;
83       focus="user";
84     //-
85     </SCRIPT>
86     <SCRIPT for="window" event="onLoad">
87     <!--
88       if (logon.user.value != "") {
89         p_retry.Start0();
90         logon.user.value = "";
91         logon.pass.value = "";
92         checkFields();
93       }
94     //-
95     </SCRIPT>
96     <!--
97       onReco="login_reco(); runSpeech0"
98       timeout="5000"
99       onInout="p_miss.Start0(); RunSpeech0 >
100      <!--
101        <script>
102          res = event.getSource();
103          if (res.text == "help") {
104            text = "復唱してください。";
105            text = document.all[focus].title;
106            p_help.Start(text);
107          }
108        </script>
109      -->
110    <BODY>
111      <!--
112        <script>
113          res = event.getSource();
114          if (res.text == "help") {
115            text = "復唱してください。";
116            text = document.all[focus].title;
117            p_help.Start(text);
118          }
119        </script>
120      -->
121    </BODY>
122  </HTML>
123
```

```

特開2003-131772 (34) 65 // 値割当てを実行する。
}
]
</script>
</head>
<body>
<prompt id="p_help" oncomplete="checkFields0()" />
<reco id="p_stock" oncomplete="stock.Start0()" /> 株式名を復唱して
<ださい、 />
<prompt id="p_op" oncomplete="op.Start0()" /> 置りですか。 黒いですか。
* </prompt>
<prompt id="p_quantity" oncomplete="quantity.Start0()" /> 同様ですか。
* </prompt>
<prompt id="p_price" oncomplete="price.Start0()" /> 価格はいくらですか。
* </prompt>
<reco id="e_stock" onReco="handle0(); checkFields0()" />
<grammar src="#/e_stock.xml" />
<reco id="e_op" onReco="handle0(); checkFields0()" />
<grammar src="#/e_op.xml" />
<reco id="e_quant" onReco="handle0(); checkFields0()" />
<grammar src="#/e_quant.xml" />
<reco id="e_price" onReco="handle0(); checkFields0()" />
<grammar src="#/e_price.xml" />
</reco>
<form id="trade">
<input name="stock" title="stock name" />
<select name="op" title="buy or sell">
<option value="buy" />
<option value="sell" />
</select>
<input name="quantity" title="number of shares" />
<input name="price" title="price" />
</form>
</body>
</html>

```

6. 2 SMLの使用
以下の例は、SMLは機構を使用したpromptおよびreco要素の活動の例を示すものである。

```

<html xmlns="urn:schemas-microsoft-com:time"
      xmlns:sp="urn:schemas-microsoft-com:speech">
<head>
<style>
<@time { behavior: url(#default#time2); }>
</style>
</head>
<body>

```

図1: smexをログインオブジェクトとして使用する

```

<smex id="logServer">
<param name="d:server" xmlns:d="urn:Microsoft.com/CDM">
<d:protocol>DCP&lt;/d:protocol>
<d:cid>209309302930202930942098432098</d:cid>

```

(37) 特開2003-131772

```

71    * .onReceive : このイベントは、プラットフォームメッセージとして使用される場合、またはこのプロトコルに null オブジェクトが割り当てる場合は無効である。
72    * .receive : 駆動専用であって、受け取ったメッセージを示す XML DOM Node データ。メッセージは、イベントが開始される前に第 1 に評価される。送信の前に、received プロパティが更新される。
73    * .onError : このイベントは、タイムアウトが発生するまで、receive として使用可能となる。
74    * .timer : 駆動ノードであって、タイムアウトイベントが発生するまでの時間で時間を示す。ミリ秒単位
75    * .bind targetElement="logServer" targetAttribute="sent" 10
76      value="#[0|log|ses|31]"
```

[0 2 4 8] この例は、クラスにはおよびインターフェース id を備えた COM オブジェクトを使用して、ログ 10 溢出に関する情報を返送するフィールドをどのように選択するかを実現したものである。音声開発者は、ロギングの関心レベルを示す属性「log」を関連する SML ノードに付け加える。上記の例では、アブリケーション開発者は、單一の bind 指示文 (directive) を使用することによって、ログ量がより大きいかまたは等しいすべてのノードをログするようになります。この例は、ダウンレベルプログラマとアップレベルプログラマが混在する。

[0 2 4 9] この例は、smex オブジェクトが既存のターフースを実施することによって、既存のポートを駆動している。同じ引数が、初数のポートを駆動するものである。同じ引数が、初数のポートを駆動する。

[0 2 5 0]

[0 2 5 1] この例は、接続呼のアドレスを読み取る

```

<smex id="telephone" sent="startListening">
  <param name="server">http://tel-srv/whatever</param>
  <bind targetElement="session_id" value="#$id#"/>
  <bind targetElement="remote" value="//remote_addr"/>
  <bind targetElement="transfer"
        value="//transfer_addr"/>
  <bind targetElement="local" value="//local_addr"/>
  ...
</smex>
```

[0 2 5 2] この例は、受け取ったメッセージを処理するためには bind 指示文を使用するかを示したものである。この例では、着信に関するメッセージがそれぞれ着信呼のリモート、転送、およびカラムアドレスを表す。サブ要素 remote_address、transfer_addr、および local_addr を有する bind 指示文に対して、上記の例は、着信呼がある場合に、固有の id が受け取られるだけであることを示している。

[0 2 5 3] 7. 1 プロパティに、HTTP 基づく接続のないプロトコルが使用される。ここで電話サーバは、複数のブラウザインスタンスと通話するように設計されたため、各クライアントは、アブリケーションの次へのページに渡すことのできるセッション ID が隠し要素として格納される。reco の場合と同様に、わからるプロトコルがリモート、転送、およびカラムアドレスを表す。サブ要素 remote_address、transfer_addr、および local_addr を有する bind 指示文が実行されるわけではなく。上記の例は、着信呼がある場合に、固有の id が受け取られるだけであることを示している。

[0 2 5 4] 7. 2 イベントオブジェクトには以下のイベントがある。

```

<smex id="logServer" ...> ...
<script>
  function logMessage(logClass, message) {
    logServer.sent = logClass + "[" + message;
  }
</script>
```

[0 2 5 5] これは、実際には、動作がカスタマイズ可能な（グローバル）関数を使用してオブジェクトを拡張するものである。上記の例では、ロギング関数は、id メッセージとの間にファイルセパレータ「|」を挿入するようになります。

[0 2 5 6] 7. 4 他の注釈

ロギング関数用に SME を拡張する約翰な方法の 1 つは、次のようになる。

```

<smex id="logServer" onload="addFunction()..."> ...
<script>
  function my_logMessage(logClass, message) {
    logServer.sent = logClass + "[" + message;
  }
</script>
```

[0 2 5 8] グローバル関数を好みない人の場合、ECMAScript の prototype プロパティを使用して、オブジェクトメソッドとして関数を付け加えることができる。たとえば、以下のような。

```

function my_logMessage(logClass, message) {
  logServer.sent = logClass + "[" + message;
}
```

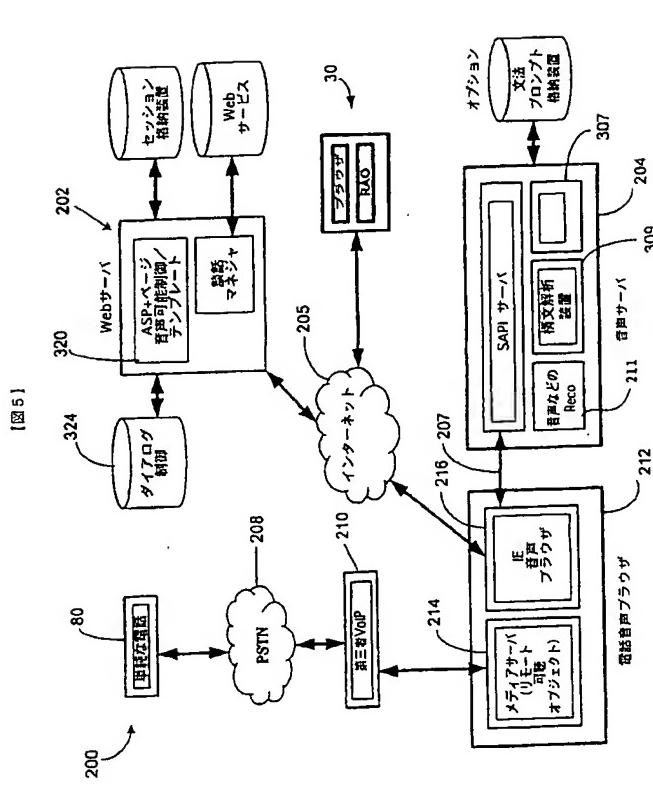
[0 2 5 9] よりオブジェクト指向の強い方法での関数は、以下のように表すことができる。

```

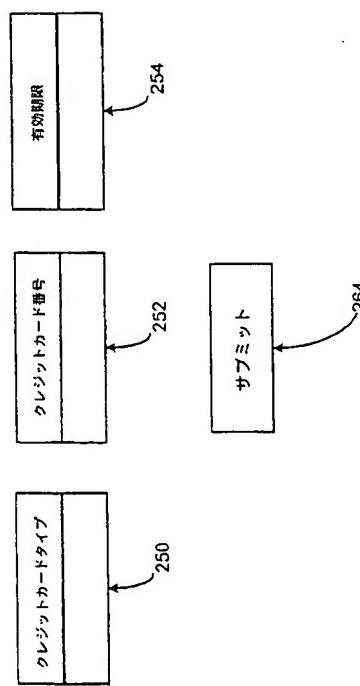
logServer.logMessage(RECLOG_ERROR, "By
  myself");
```

[0 2 6 0] すべての必要な機能はすでに標準で構立さ

50 [図] コンピュータインターフェイスの動作環境の第 1



[図6]



[図7]

```

(41) 特開2003-131772

[図7]

<html>
<form id="get_card_info" method="post" action="http://paymant.asp">
    <select name="card_type">
        <option value="amex">American Express</option>
        <option value="visa">Visa</option>
    </select>
    <input type="text" name="card_num" width="70" onchange="MasterCard()"/>
    <input type="text" name="expiry_date" />
    <input type="submit" value="Submit" onclick="Verify()"/>
</form>
<script></script>

```

260

```

function handle() {
    if (get_card_info.card_type.value == "amex") {
        if (get_card_info.info.card_num.length != 15)
            alert ("Amex should have 15 digits");
        else if (get_card_info.info.card_num.length != 16)
            alert ("Visa and master should have 16 digits");
    }
}

```

262

```

function verify() {
    var flag = window.confirm("submit the credit card info?");
    if (flag)
        get_card_info.submit();
}

```

30

```

</script>
</html>

```

[図10]

```

function handleO {
    handle();
    checkP11();
}

function checkP11edO {
    if (card_type.value == "") {
        p_card_type.activeO = false;
        g_card_types.activateO = return;
    }
    if (card_num.value == "") {
        p_card_num.activeO = false;
        g_card_num.activateO = return;
    }
    if (expiry_date.value == "") {
        p_expiry_date.activeO = false;
        g_expiry_date.activateO = return;
    }
    p_content.activateO();
    p_confirm.activateO();
    confirmation.activateO();
}

function confirmed(gobj) {
    if (gobj.recognize.text == "yes")
        get_card_info.submit();
}

// user codes start here

```

324

```

if (get_card_info.card_type == "amex") {
    if (get_card_info.info.card_num.length != 15)
        prompt_speak ("Amex should have 15 digits");
    get_card_info.card_num = "";
}
else if (get_card_info.info.card_num.length != 16) {
    prompt_speak ("Visa and master should have 16 digits");
}

```

254

```

get_card_info.submit();
}

```

264

```

</script>
</body>

```

```

<html>
  <form id="get_card_info" method="post" action="http://Payment.aspx">
    <select name="card_type" onclick="talk(0_card_type)">
      <option value="ame"> American Express </option>
      <option value="mf"> MasterCard </option>
    </select>

    <recoid="g_card_types" onRecId="handle" >
      <grammar src="/Gram/credit_cards.gsrc" type="text/x-speech" width="30" />
      <rule name="card_num" handle="handle1" >
        <grammar src="g_card_num.gram" type="text/x-speech" digits="16" />
        <rule name="expiry_date" handle="handle2" >
          <grammar src="g_expiry_date.gram" type="text/x-speech" />
        <rule name="submit" handle="handle3" >
          <grammar src="submit.gram" type="text/x-speech" />
        </rule>
      </rule>
    </recoid>
    <input type="text" name="expiry_date" onclick="talk(0_expiry_date)">
    <recoid="g_expiry_date" >
      <grammar src="g_expiry_date.gram" type="text/x-speech" />
    </recoid>
    <input type="submit" value="Submit" onClick="verify()"/>
  </form>
  <script>
    function talk(node) {
      gobj.activetext();
    }
    function handle1() {
      if (get_card_info.card_num != null) {
        if (get_card_info.card_type.value == "amex") {
          if (get_card_info.card_num.length == 15)
            alert("Amex should have 15 digits");
        } else {
          if (get_card_info.card_num.length == 16)
            alert("Visa and master should have 16 digits");
        }
      }
    }
    function verify() {
      var flag = window.confirm("submit the credit card info?");
      if (flag) {
        get_card_info.submit();
      }
    }
  </script>

```

```

  <body>
    <form id="get-card-info" method="post" action="http://permaatastic">
      <input type="text" value="welcome0" />
      <input type="text" value="I have a credit card" />
      <input type="text" value="I didn't understand you" />
      <input type="text" value="What credit card would you like?" />
      <input type="text" value="Please say the number" />
      <input type="text" value="P-CRDT-NUM" />
      <input type="text" value="Bargin=true" />
      <input type="text" value="W is the expiration date?" />
      <input type="text" value="I have your value select="card_num" />
      <input type="text" value="with expiration date" />
      <input type="text" value="select="expiry_date" />
      </form>
      <pre>
        <?xml version="1.0"?>
        <response>
          <error>0</error>
          <result>
            <id>1</id>
            <type>success</type>
            <status>true</status>
            <msg>What credit card would you like?</msg>
            <msg>Please say the number</msg>
            <msg>P-CRDT-NUM</msg>
            <msg>Bargin=true</msg>
            <msg>W is the expiration date?</msg>
            <msg>I have your value select="card_num" with expiration date</msg>
            <msg>select="expiry_date" /</msg>
          </result>
        </response>
      </pre>
      300
      303
      305
    </body>
  
```

10

(43)

図10～

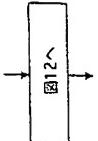
(44) 特別2003-131772

[図1.1]

```

<body>
<form id="get_card_info" method="post" action="http://payment.asp"
onactivate="welcome();"
onprompt id="p_welcome">We now need your credit card</prompt>
<prompt id="p_mumble">I didn't understand >What credit card would you
use?</prompt>
<prompt id="p_card_type" value="true">What credit card would you
use?</prompt>
<prompt id="p_card_num" value="true">Please say the number</prompt>
<prompt id="p_expiry_date" value="true">What is the expiration
date?</prompt>
<prompt id="p_content">
I have your <value select="card_type"/><value select="card_num"/>
with expiration date <value select="expiry_date"/>
</prompt>
<prompt id="p_confirm">Is this correct?</prompt>
<reco id="get_card_info" onHandle="handle()" onNoReco="mumble(this)">
<reco id="get_card_info" onHandle="handle()" onNoReco="mumble(this)">
<grammar src="~/tran/payment.gram" />
<bind target="card_type" value="~/card/type" />
<bind target="card_num" value="~/card/num" />
<bind target="expiry_date" value="~/card/expir_date" />
<reco>
<reco id="confirmation" onReco = "confirmed(this)" onNoReco = "mumble(this)" />
<reco>
<select name="card_type">
<option value="amex">American Express</option>
<option value="visa">Visa</option>
<option value="ms">MasterCard</option>
<select>
<input type="text" name="card_num" width="30" />
<input type="text" name="expiry_date" />
<input type="submit" value="Submit" />
</form>
<script>(function() {
    function deactivate(obj) {
        obj.deactivate();
        checkIfId();
        deactivate();
    }
    function activate(obj) {
        obj.activate();
        checkIfId();
        activate();
    }
})()
</script>
</body>

```



(45) 特別2003-131772

[図1.2]

```

function checkFilled() {
    if (card_type.value == "") {
        p_card_type.activate();
        return;
    }
    if (card_num.value == "") {
        p_card_num.activate();
        return;
    }
    if (expiry_date.value == "") {
        p_expiry_date.activate();
        return;
    }
    p_content.activate();
    p_confirm.activate();
    confirmation.activate();
}

function confirmed(gobj) {
    if (gobj.value == "yes") {
        get_card_info.submit(gensei());
    }
}

// user codes start here
function handle() {
    if (field == get_card_info.card_num) {
        if (get_card_info.card_num.length == 15) {
            prompt.speak("Name Shou o have 15 digits");
            get_card_info.card_num = "";
        } else {
            if (get_card_info.card_num.length != 16) {
                get_card_info.card_num = "Name Shou o have 15 digits";
            }
        }
    }
}

function gensei() {
    str = "<com><credit_card type='>" + field + "<number>" + card_number.value + str + "</number><expires>" +
    str + "<card_type>" + card_type.value + str + "</card_type>" +
    str + "<card_number>" + card_number.value + str + "</card_number>" +
    str + "<expiry_date>" + expiry_date.value + str + "</expiry_date>" +
    str + "<credit_card>" + card_number.value + str + "</credit_card>" +
    str + "</com></credit_card>" + str;
    return str;
}

```

特開2003-131772

(46) SAMPLE PAGE

```

<! Page language="JavaScript" AutoEventWireup="True" Inherits="Credit.Transaction" %>
<!-- ASPX page for both voice-only & multichannel credit card example -->
<script>
    //...
    function handle() {
        if (card_id == get_card_info(card_num)) {
            if (get_card_info("Card Num Length") != 15) {
                browser_speak("Your card number should have 15 digits!");
                go_to_card_info("Card Num");
            } else {
                if (user_card_info.card_num.length != 16) {
                    browser_speak("This card number should have 15 digits!");
                    go_to_card_info("Card Num");
                }
            }
        }
    }

    function generate() {
        str = "<html><credit card><card type='>" + 
        str += "</card><card value='>" + 
        str += "</card><type><number>" + 
        str += "</number><expdate></expdate></credit></html>";
        return str;
    }

    function validate() {
        if (PostBack) {
            validation = new System.Speech.SynthesisValidator("./CreditSDL.xml");
            deal = validation.Validate(args);
            deal = validate_deal(deal);
            if (!deal) {
                browser_speak("Please choose a deal!");
            } else {
                // Initialize fields with args
            }
        }
    }

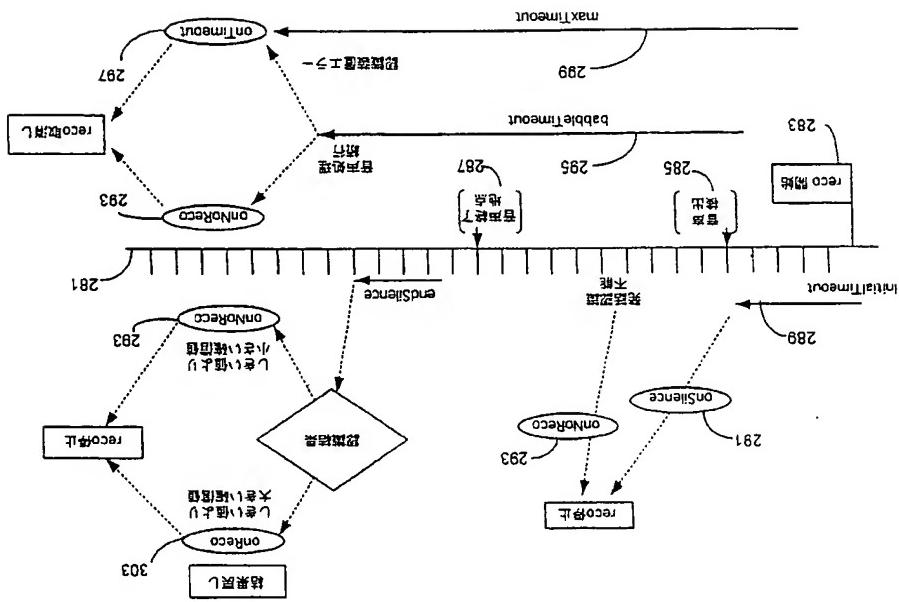
    <script runat="server">
        <!-- Page_Load (obj, args) -->
        Function Page_Load (obj, args) {
            validate();
            if (PostBack) {
                validation = new System.Speech.SynthesisValidator("./CreditSDL.xml");
                deal = validation.Validate(args);
                if (!deal) {
                    browser_speak("Please choose a deal!");
                } else {
                    // Initialize fields with args
                }
            }
        }
    </script>
</head>
<body>
    <asp:Form id="GetCardInfo" style="display: inline-block;" OnLoad="SystemInitiate">
        <asp:Speech:Choice Name="Card Type" Prompt="What credit card would you use?">
            <asp:Grammar Name="Card Type" Prompt="What credit card would you use?">
                <option>American Express</option>
                <option>Diners</option>
                <option>MasterCard</option>
            </asp:Grammar>
            <asp:Textbox Name="Card Number" Prompt="Please say the number">
                <asp:Grammar>[0-9]{16}</asp:Grammar>
            </asp:Textbox>
            <asp:Textbox Name="Expiry Date" Prompt="What is the expiration date?">
                <asp:Grammar>[0-9]{2}/[0-9]{2}</asp:Grammar>
            </asp:Textbox>
        </asp:Speech:Choice>
    </asp:Form>
</body>
</html>

```

[図1.3]

(47)

[図1.4]



特開2003-131772

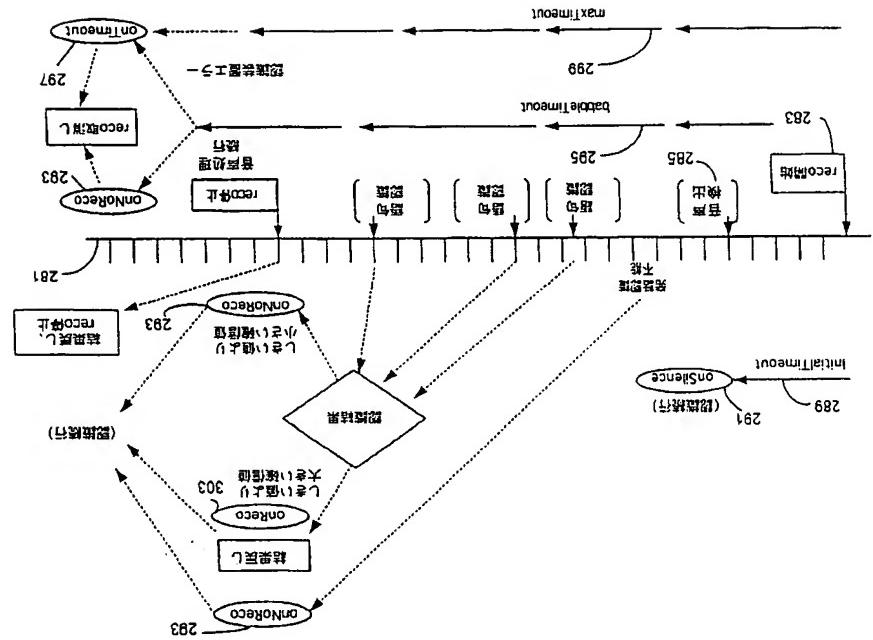
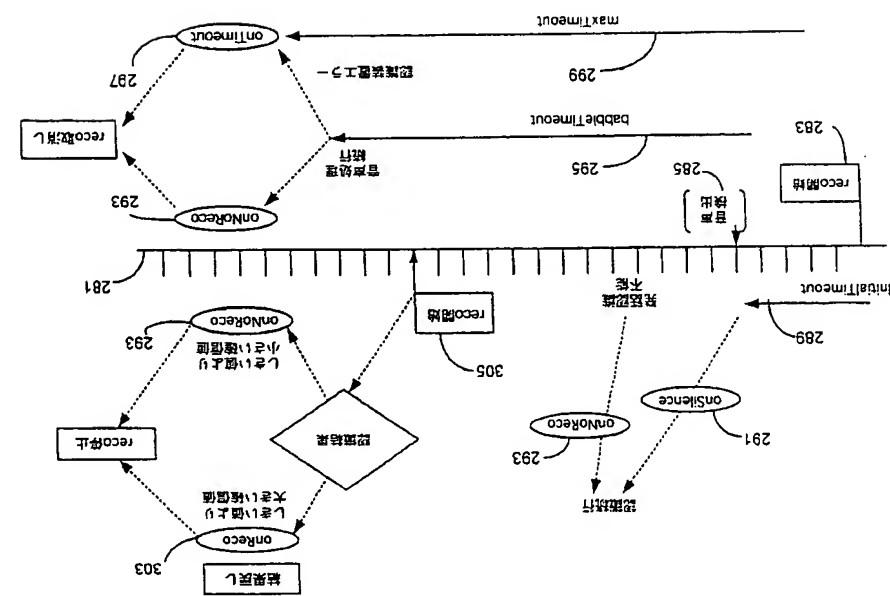


图 17

18]

18]

フロントページの継ぎ

(72)発明者	ウアン クアンサン アメリカ合衆国	98006 ワシントン州 ベルヒュー サウスイースト 48 コート 16470	(72)発明者: ホン シヤオーワーハン アメリカ合衆国 98006 ワシントン州 ベルヒュー サウスイースト 58 ブレイ ス 17797
F ターム(参考)	5D015 KW03		
SE501	AA03 AD04 AB03 BA05 BA11		
GB04	CB05 CB06 CB08 CB09		
GB10	CB13 CB15 CC17 EK21		
FA04			

(72) 発明者 ホン シャオウエン

KK03 AA03 AA04 AB03 BA05 BA11
CB04 CB05 CB06 CB08 CB09
CB10 CB13 CB15 CC17 EA21
FA04

Z 17797

F タ — L (参考)	5D015 K03	5E501 A03 AA04 AB03 BA05 BA11
	CB04 CB05 CB06 CB08 CB09	
	CB10 CB13 CB15 CC17 EA21	
	F0A4	

۱۶۴

【技術分類】

1. Title of the Invention

**MARUP LANGUAGE EXTENSIONS FOR WEB ENABLED
RECOGNITION**

2. Claims

1. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating a grammar to associate with input entered through the client device.
 2. The computer readable medium of claim 1 wherein the instruction indicating a grammar provides a reference to a location of the grammar.
 3. The computer readable medium of claim 1 wherein the grammar is for one of speech recognition, handwriting recognition, gesture recognition and visual recognition.
 4. The computer readable medium of claim 3 wherein the instruction indicating a grammar includes a reference to a grammar for speech recognition.
 5. The computer readable medium of claim 1 wherein the markup language includes an instruction for receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.
 6. The computer readable medium of claim 8 wherein the instruction for receiving a recognition result associates the recognition result to a plurality of data fields.
7. The computer readable medium of claim 1 wherein the markup language includes an instruction to audibly render an indication when the speech was not recognized.
8. The computer readable medium of claim 1 wherein the input comprises a text string.
9. The computer readable medium of claim 8 wherein the markup language includes an instruction for receiving the text string from another component.
10. The computer readable medium of claim 9 wherein markup language associates the grammar to process the text string.
11. A computer readable medium including instructions readable by a computer which, when implemented, cause the computer to handle information by performing steps comprising:
 - receiving a markup language page from a web server having a field for input data;
 - receiving input from a user related to the field; and
 - sending data indicative of the input and an indication of a grammar for recognition.
12. The computer readable medium of claim 11 wherein the indication provides a reference to locations of the grammars.
13. The computer readable medium of claim 12 wherein the indication includes a reference to a language for speech recognition.

14. A method for recognition in a client/server network, the method comprising:
receiving a markup language page having a field for input data on a client device connected to a network from a web server at an address on the network different than the client device;
receiving input from a user related to the field on the client device; and
sending data indicative of the input and an indication of a grammar for recognition to a recognition server at an address on the network different than the client device.
15. The method of claim 14 and further comprising:
normalizing the data indicative of the input prior to sending the data.
16. The method of claim 14 and further comprising:
receiving a recognition result pertaining to recognized speech and associating the result with a data field on the client device.
17. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for recognition.
18. The computer readable medium of claim 17 wherein the element is for one of speech recognition, handwriting recognition, gesture recognition, DTMF recognition and visual recognition.
19. The computer readable medium of claim 17 wherein the element comprises an indication of a grammar for recognition.
20. The computer readable medium of claim 19 wherein the element pertains to receiving and processing a text string like recognition.
21. The computer readable medium of claim 20 and further comprising a second object model receiving the text string from another component.
22. A computer readable medium having a markup language for execution on a client device in a client/server system, the markup language comprising an instruction indicating an object model element having an attribute or a method for audibly prompting.
23. The computer readable medium of claim 22 wherein the markup language comprises one of HTML, XHTML, XML, and WML.
24. The computer readable medium of claim 22 wherein the markup language comprises a scripting language.
25. The computer readable medium of claim 22 wherein the markup language comprises a synchronized markup language.

3. Detailed Description of the Invention**FIELD OF THE INVENTION**

The present invention relates to access of information over a wide area network such as the Internet. More particularly, the present invention relates to web enabled recognition allowing information and control on a client side to be entered using a variety of methods.

BACKGROUND OF THE INVENTION

Small computing devices such as personal information managers (PIM), devices and portable phones are used with ever increasing frequency by people in their day-to-day activities. With the increase in processing power now available for microprocessors used to run these devices, the functionality of these devices are increasing, and in some cases, merging. For instance, many portable phones now can be used to access and browse the Internet as well as can be used to store personal information such as addresses, phone numbers and the like.

In view that these computing devices are being used for browsing the Internet, or are used in other server/client architectures, it is therefore necessary to enter information into the computing device. Unfortunately, due to the desire to keep these devices as small as possible in order that they are easily carried, conventional keyboards having all the letters of the alphabet as isolated buttons are usually not possible due to the limited surface area available on the housings of the computing devices.

Recently, voice portals such as through the use of VoiceXML documents in reply, which are processed by the VoiceXML interpreter and rendered audibly to the user. Using voice commands through voice recognition, the user can navigate the web.

example, a web server) processes requests from a client through a VoiceXML interpreter. The web server can produce VoiceXML documents in reply, which are processed by the VoiceXML interpreter and rendered audibly to the user. Using voice commands through voice recognition, the user can navigate the web.

VoiceXML is a markup language with flow control tags; however, flow control does not follow the HTML (Hyper Text Markup Language) flow control model, which includes eventing and separate scripts. Rather, VoiceXML generally includes a form interpretation algorithm that is particularly suited for telephone-based voice-only interaction, and commonly, where the information obtained from the user is under the control of the system or application. Incorporation of VoiceXML directly into applications available in a client-server relationship where graphically user interfaces are also provided will require the developer to master two forms of web authoring, one for VoiceXML, and the other using HTML (or the like), each one following a different flow control model. There is thus an ongoing need to improve upon the architecture and methods used to provide speech recognition in a server/client architecture such as the Internet. The authoring tool for speech recognition should be easily adaptable to small computing devices such as PIMs, telephones and the like. An architecture or method of web authoring that addresses one, several or all of the foregoing disadvantages is particularly needed.

Recently, voice extensible markup language have been advanced to allow Internet content to be accessed using only a telephone. In this architecture, a document server (for

SUMMARY OF THE INVENTION

A markup language for execution on a client device in a client/server system includes an instruction indicating a grammar to associate with input data entered through the client device.

With the availability of this extension and as another aspect of the present invention, a client device can execute instructions to receive a markup language page from a web server having a field for input data. The client device can then receive input data from a user related to the field and send the data and an indication of the grammar for recognition to a recognition server, typically, located at a remote location for processing.

The recognition server can execute instructions to receive the input data and the indication of the grammar to perform recognition. The results of recognition can then be sent back to the client device or web server for further processing.

DETAILED DESCRIPTION OF THE INVENTION

Before describing an architecture of web based recognition and methods for implementing the same, it may be useful to describe generally computing devices that can function in the architecture. Referring now to FIG. 1, an exemplary form of a data management device (PIM, PDA or the like) is illustrated at 30. However, it is contemplated that the present invention can also be practiced using other computing devices discussed below, and in particular, those computing devices having limited surface areas for input buttons or the like. For example, phones and/or data management devices will also benefit from the present invention. Such devices will have an enhanced utility compared to existing portable personal information management devices and other portable electronic devices, and the functions and compact size of such devices will more likely encourage the user to carry the device at all times. Accordingly, it is not intended that the scope of the architecture herein described be limited by the disclosure of

an exemplary data management or PIM device, phone or computer herein illustrated.

An exemplary form of a data management mobile device 30 is illustrated in FIG. 1. The mobile device 30 includes a housing 32 and has an user interface including a display 34, which uses a contact sensitive display screen in conjunction with a stylus 33. The stylus 33 is used to press or contact the display 34 at designated coordinates to select a field, to selectively move a starting position of a cursor, or to otherwise provide command information such as through gestures or handwriting. Alternatively, or in addition, one or more buttons 35 can be included on the device 30 for navigation. In addition, other input mechanisms such as rotatable wheels, rollers or the like can also be provided. However, it should be noted that the invention is not intended to be limited by these forms of input mechanisms. For instance, another form of input can include a visual input such as through computer vision.

Referring now to FIG. 2, a block diagram illustrates the functional components comprising the mobile device 30. A central processing unit (CPU) 50 implements the software control functions. CPU 50 is coupled to display 34 so that text and graphic icons generated in accordance with the controlling software appear on the display 34. A speaker 43 can be coupled to CPU 50 typically with a digital-to-analog converter 59 to provide an audible output. Data that is downloaded or entered by the user into the mobile device 30 is stored in a non-volatile random access memory store 54 bi-directionally coupled to the CPU 50. Random access memory (RAM) 54 provides volatile storage for instructions that are executed by CPU 50, and storage for temporary data, such as register values. Default values for

(61) configuration options and other variables are stored in a read only memory (ROM) 58. ROM 58 can also be used to store the operating system software for the device that controls the basic functionality of the mobile 30 and other operating system kernel functions (e.g., the loading of software components into RAM 54).

RAM 54 also serves as a storage for the code in the manner analogous to the function of a hard drive on a PC that is used to store application programs. It should be noted that although non-volatile memory is used for storing the code, it alternatively can be stored in volatile memory that is not used for execution of the code.

Wireless signals can be transmitted/received by the mobile device through a wireless transceiver 52, which is coupled to CPU 50. An optional communication interface 60 can also be provided for downloading data directly from a computer (e.g., desktop computer), or from a wired network, if desired. Accordingly, interface 60 can comprise various forms of communication devices, for example, an infrared link, modem, a network card, or the like.

Mobile device 30 includes a microphone 29, and analog-to-digital (A/D) converter 37, and an optional recognition program (speech, DTMF, handwriting, gesture or computer vision) stored in store 54. By way of example, in response to audible information, instructions or commands from a user of device 30, microphone 29 provides speech signals, which are digitized by A/D converter 37. The speech recognition program can perform normalization and/or feature extraction functions on the digitized speech signals to obtain intermediate speech recognition results. Using wireless transceiver 52 or communication interface 60, speech data is transmitted to a remote recognition server 204 discussed below and illustrated

in the architecture of FIG. 5. Recognition results are then returned to mobile device 30 for rendering (e.g., visual and/or audible) thereon, and eventual transmission to a web server 202 (FIG. 5), wherein the web server 202 and mobile device 30 operate in a client/server relationship. Similar processing can be used for other forms of input. For example, handwriting input can be digitized with or without pre-handwriting processing on device 30. Like the speech data, this form of input can be transmitted to the recognition server 204 for recognition wherein the recognition results are returned to at least one of the device 30 and/or web server 202. Likewise, DTMF data, gesture data and visual data can be processed similarly. Depending on the form of input, device 30 (and the other forms of clients discussed below) would include necessary hardware such as a camera for visual input.

FIG. 3 is a plan view of an exemplary embodiment of a portable phone 80. The phone 80 includes a display 82 and a keypad 84. Generally, the block diagram of FIG. 2 applies to the phone of FIG. 3, although additional circuitry necessary to perform other functions may be required. For instance, a transceiver necessary to operate as a phone will be required for the embodiment of FIG. 2; however, such circuitry is not pertinent to the present invention.

In addition to the portable or mobile computing devices described above, it should also be understood that the present invention can be used with numerous other computing devices such as a general desktop computer. For instance, the present invention will allow a user with limited physical abilities to input or enter text into a computer or other computing device when other conventional input devices, such as a full alpha-numeric keyboard, are too difficult to operate.

The invention is also operational with numerous other general purpose or special purpose computing systems, environments or configurations. Examples of well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but are not limited to, regular telephones (without any screen) personal computers, server computers, hand-held or laptop devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronics, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

The following is a brief description of a general-purpose computer 120 illustrated in FIG. 4. However, the computer 120 is again only one example of a suitable computing environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Neither should the computer 120 be interpreted as having any dependency or requirement relating to any one or combination of components illustrated therein.

The invention may be described in the general context of computer-executable instructions, such as program modules, being executed by a computer. Generally, program modules include routines, programs, objects, components, data structures, etc. that perform particular tasks or implement particular abstract data types. The invention may also be practiced in distributed computing environments where tasks are performed by remote processing devices that are linked through a communications network. In a distributed computing environment, program modules may be located in both local and remote computer storage media including memory storage devices. Tasks performed by the

programs and modules are described below and with the sic of figures. Those skilled in the art can implement the description and figures as processor executable instructions, which can be written on any form of a computer readable medium.

With reference to FIG. 4, components of computer 120 may include, but are not limited to, a processing unit 140, a system memory 150, and a system bus 141 that couples various system components including the system memory to the processing unit 140. The system bus 141 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. By way of example, and not limitation, such architectures include Industry Standard Architecture (ISA) bus, Universal Serial Bus (USB), Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus. Computer 120 typically includes a variety of computer readable mediums. Computer readable mediums can be any available media that can be accessed by computer 120 and includes both volatile and nonvolatile media, removable and non-removable media. By way of example, and not limitation, computer readable mediums may comprise computer storage media and communication media. Computer storage media includes both volatile and nonvolatile, removable and non-removable media implemented in any method or technology for storage of information such as computer readable instructions, data structures, program modules or other data. Computer storage media includes, but is not limited to, RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital

versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium which can be used to store the desired information and which can be accessed by computer 120.

Communication media typically embodies computer readable instructions, data structures, program modules or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. By way of example, and not limitation, communication media includes wired media such as a wired network or direct-wired connection, and wireless media such as acoustic, FR, infrared and other wireless media. Combinations of any of the above should also be included within the scope of computer readable media.

The system memory 150 includes computer storage media in the form of volatile and/or nonvolatile memory such as read only memory (ROM) 151 and random access memory (RAM) 152. A basic input/output system 153 (BIOS), containing the basic routines that help to transfer information between elements within computer 120, such as during start-up, is typically stored in ROM 151. RAM 152 typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processing unit 140. By way of example, and not limitation, FIG. 4 illustrates operating system 54, application programs 155, other program modules 156, and program data 157.

The computer 120 may also include other removable/non-removable volatile/nonvolatile computer storage media. By way of example only, FIG. 4 illustrates a hard disk drive 161 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 171 that reads from or writes to a removable, nonvolatile magnetic disk 172, and an optical disk drive 175 that reads from or writes to a removable, nonvolatile optical disk 176 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 161 is typically connected to the system bus 141 through a non-removable memory interface such as interface 160, and magnetic disk drive 171 and optical disk drive 175 are typically connected to the system bus 141 by a removable memory interface, such as interface 170.

The drives and their associated computer storage media discussed above and illustrated in FIG. 4, provide storage of computer readable instructions, data structures, program modules and other data for the computer 120. In FIG. 4, for example, hard disk drive 161 is illustrated as storing operating system 164, application programs 165, other program modules 166, and program data 167. Note that these components can either be the same as or different from operating system 154, application programs 155, other program modules 156, and program data 157. Operating system 164, application programs 165, other program modules 166,

and program data 167 are given different numbers here to illustrate that, at a minimum, they are different copies.

A user may enter commands and information into the computer 120 through input devices such as a keyboard -82, a microphone 183, and a pointing device 181, such as a mouse, trackball or touch pad. Other input devices (not shown) may include a joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 140 through a user input interface 180 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 184 or other type of display device is also connected to the system bus 141 via an interface, such as a video interface 185. In addition to the monitor, computers may also include other peripheral output devices such as speakers 187 and printer 186, which may be connected through an output peripheral interface 188.

The computer 120 may operate in a networked environment using logical connections to one or more remote computers, such as a remote computer 134. The remote computer 194 may be a personal computer, a hand-held device, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described above relative to the computer 120. The logical connections depicted in FIG. 4 include a local area network (LAN) 191 and a wide area network (WAN) 193, but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

(68) When used in a LAN networking environment, the computer 120 is connected to the LAN 191 through a network interface or adapter 190. When used in a WAN networking environment, the computer 120 typically includes a modem 192 or other means for establishing communications over the WAN 193, such as the Internet. The modem 192, which may be internal or external, may be connected to the system bus 141 via the user input interface 180, or other appropriate mechanism. In a networked environment, program modules depicted relative to the computer 120, or portions thereof, may be stored in the remote memory storage device. By way of example, and not limitation, FIG. 4 illustrates remote application programs 195 as residing on remote computer 194. It will be appreciated that the network connections shown are exemplary and other means of establishing a communications link between the computers may be used.

FIG. 5 illustrates architecture 200 for web based recognition as can be embodied in the present invention. Generally, information stored in a web server 202 can be accessed through mobile device 30 (which herein also represents other forms of computing devices having a display screen, a microphone, a camera, a touch sensitive panel, etc., as required based on the form of input), or through phone 80 wherein information is requested audibly or through tones generated by phone 80 in response to keys depressed and wherein information from web server 202 is provided only audibly back to the user.

More importantly though, architecture 200 is unified in that whether information is obtained through device 30 or phone 80 using speech recognition, a single recognition server 204 can support either mode of operation. In addition, architecture 200 operates using an

(69) extension of well-known mark-up languages (e.g., HTML, XHTML, cHTML, XML, WML, and the like). Thus, information stored on web server 202 can also be accessed using well-known GUI methods found in these mark-up languages. By using an extension of well-known mark-up languages, authoring on the web server 202 is easier, and legacy applications currently existing can be also easily modified to include voice recognition.

Generally, device 30 executes HTML pages, scripts, or the like, provided by web server 202. When voice recognition is required, by way of example, speech data, which can be digitized audio signals or speech features wherein the audio signals have been preprocessed by device 30 as discussed above, are provided to recognition server 204 with an indication of a grammar or language model to use during speech recognition. The implementation of the recognition server 204 can take many forms, one of which is illustrated, but generally includes a recognizer 211. The results of recognition are provided back to device 30 for local rendering if desired or appropriate. Upon compilation of information through recognition and any graphical user interface if used, device 30 sends the information to web server 202 for further processing and receipt of further HTML pages/scripts, if necessary.

As illustrated in FIG. 5, device 30, web server 202 and recognition server 204 are commonly connected, and separately addressable, through a network 205, herein a wide area network such as the Internet. It therefore is not necessary that any of these devices be physically located adjacent each other. In particular, it is not necessary that web server 202 includes recognition server 204. In

this manner, authoring at web server 202 can be focused on the application to which it is intended without the authors needing to know the intricacies of recognition server 204. Rather, recognition server 204 can be independently designed and connected to the network 205, and thereby, be updated and improved without further changes required at web server 202. As discussed below, web server 202 can also include an authoring mechanism that can dynamically generate client-side markups and scripts. In a further embodiment, the web server 202, recognition server 204 and client 30 may be combined depending on the capabilities of the implementing machines. For instance, if the client comprises a general purpose computer, e.g. a personal computer, the client may include the recognition server 204. Likewise, if desired, the web server 202 and recognition server 204 can be incorporated into a single machine.

An aspect of the present invention is a method for processing input data in a client/server system that includes receiving from a server a markup language page having extensions configured to obtain input data from a user of a client device; executing the markup language page on the client device transmitting input data (indicative of speech, DTMF, handwriting, gestures or images obtained from the user) and an associated grammar to a recognition server remote from the client; and receiving a recognition result from the recognition server at the client. Another aspect is a computer readable medium having a markup language for execution on a client device in a client/server system, the markup language having an instruction indicating a grammar to associate with input data entered through the client device.

Access to web server 202 through phone 80 includes connection of phone 80 to a wired or wireless telephone network 208, that in turn, connects phone 80 to a third party gateway 210. Gateway 210 connects phone 80 to a telephony voice browser 212. Telephony voice browser 212 includes a media server 214 that provides a telephony interface and a voice browser 216. Like device 30, telephony voice browser 212 receives HTML pages/scripts or the like from web server 202. More importantly though, the HTML pages/scripts are of the form similar to HTML pages/scripts provided to device 30. In this manner, web server 202 need not support device 30 and phone 80 separately, or even support standard GUI clients separately. Rather, a common mark-up language can be used. In addition, like device 30, voice recognition from audible signals transmitted by phone 80 are provided from voice browser 216 to recognition server 204, either through the network 205, or through a dedicated line 207, for example, using TCP/IP. Web server 202, recognition server 204 and telephone voice browser 212 can be embodied in any suitable computing environment such as the general purpose desktop computer illustrated in FIG. 4.

However, it should be noted that if DTMF recognition is employed, this form of recognition would generally be performed at the media server 214, rather than at the recognition server 204. In other words, the DTMF grammar would be used by the media server.

As indicated above, one aspect of the present invention includes extension of mark-up languages such as HTML, XHTML, XML, WML or with any other SGML-derived markup to include controls and/or objects that provide recognition in a client/server architecture. In this

(72) manner, authors can leverage all the tools and expertise in these mark-up languages that are the predominant web development platform used in such architectures.

Generally, controls and/or objects can include one or more of the following functions: recognizer controls and/or objects for recognizer configuration, recognizer execution and/or post-processing; synthesizer controls and/or objects for synthesizer configuration and prompt playing; grammar controls and/or objects for specifying input grammar resources; and/or binding controls and/or objects for processing recognition results. The extensions are designed to be a lightweight markup layer, which adds the power of an audible, visual, handwriting, etc. interface to existing markup languages. As such, the extensions can remain independent of: the high-level page in which they are contained, e.g. HTML; the low-level formats which the extensions used to refer to linguistic resources, e.g. the text-to-speech and grammar formats; and the individual properties of the recognition and speech synthesis platforms used in the recognition server 204.

Before describing mark-up languages having controls and/or objects suited for recognition, it may be helpful to examine a simple GUI example herein embodied, with the HTML mark-up language. Referring to FIG. 6, a simple GUI interface comprises submission of credit card information to the web server to complete an on-line sale. In this example, the credit card information includes a field 250 for entry of the type of credit card being used, for example, Visa, MasterCard or American Express. A second field 252 allows entry of the credit card number, while a third field 254 allows entry of the expiration date. Submit button 264 is provided to transmit the information entered in fields 250, 252 and 254.

FIG. 7 illustrates the HTML code for obtaining the foregoing credit card information from the client.

Generally, as is common in these forms of markup languages, the code includes a body portion 260 and a script portion 262. The body portion 260 includes lines of

script indicating the type of action to be performed, the code to use, the various fields of information 250, 252 and 254, as well as a code for submit button 264 (FIG. 6). This example also illustrates exerting support and embedded script hosting, wherein upon activation of the submit button 264, a function "verify" is called or executed in script portion 262. The "verify" function ascertains whether the card number length for each of the credit cards (Visa, MasterCard and American Express) is of the proper length.

FIG. 8 illustrates a client markup that generates the same GUI of FIG. 6 for obtaining credit card information to be provided to web server 204 using speech recognition. Although speech recognition will be discussed below with respect to FIGS. 8-14, it should be understood that the techniques described can be similarly applied in handwriting recognition, gesture recognition and image recognition.

Generally, the extensions (also commonly known as "tags") are a small set of XML elements, with associated attributes and DOM object properties, events and methods, which may be used in conjunction with a source markup document to apply a recognition interface, DTMF or call control to a source page. The extensions formalities and semantics are independent of the nature of the source

document, so the extensions can be used equally effectively within HTML, XHTML, XML, WML, or with any other SGML-derived markup. The extension follow the document object model wherein new functional objects or elements, which can be hierarchical, are provided. Each of the elements are discussed in detail in the Appendix, but generally the elements can include attributes, properties, methods, events and/or other "child" elements.

At this point, it should also be noted that the extensions may be interpreted in two different "modes" according to the capabilities of the device upon which the browser is being executed on. In a first mode, "object mode", the full capabilities are available. The programmatic manipulation of the extensions by an application is performed by whatever mechanisms are enabled by the browser on the device, e.g. a JavaScript interpreter in an XHTML browser, or a WMLScript interpreter in a WML browser. For this reason, only a small set of core properties and methods of the extensions need to be defined, and these manipulated by whatever programmatic mechanisms exist on the device or client side. The object mode provides eventing and scripting and can offer greater functionality to give the dialog author a much finer client-side control over speech interactions. As used herein, a browser that supports full event and scripting is called an "uplevel browser". This form of a browser will support all the attributes, properties, methods and events of the extensions. Uplevel browsers are commonly found on devices with greater processing capabilities.

The extensions can also be supported in a

"declarative mode". As used herein, a browser operating in

a declarative mode is called a "downlevel browser" and does

not support full eventing and scripting capabilities. Rather, this form of browser will support the declarative aspects of a given extension (i.e. the core element and attributes), but possibly not all the DOM (document object model) object properties, methods and events. This mode employs exclusively declarative syntax, and may further be used in conjunction with declarative multimedia synchronization and coordination mechanisms (synchronized multimedia markup language) such as SMIL (Synchronized Multimedia Integration Language) 2.0. Downlevel browsers will typically be found on devices with limited processing capabilities.

At this point though, a particular mode of entry should be discussed. In particular, use of speech recognition in conjunction with at least a display and, in a further embodiment, a pointing device as well to indicate the fields for data entry is particularly useful. Specifically, in this mode of data entry, the user is generally under control of when to select a field and provide corresponding information. For instance, in the example of FIG. 6, a user could first decide to enter the credit card number in field 252 and then enter the type of credit card in field 250 followed by the expiration date in field 254. Likewise, the user could return back to field 252 and correct an errant entry, if desired. When combined with speech recognition as described below, an easy and natural form of navigation is provided. As used herein, this form of entry using both a screen display allowing free form selection of fields and voice recognition is called "multi-modal".

Referring back to FIG. 8, an example of HTML mark-up language code is illustrated. Like the HTML code

further embodiment, information corresponding to expected input to be entered, for example, in a specific field) A new control 290 (herein identified as "reco"), comprising a first extensor of the mark-up language, includes various elements, two of which are illustrated, namely a grammar element "grammar" and a "bind" element. Generally, like the code downloaded to a client from web server 202, the grammars can originate at web server 202 and be downloaded to the client and/or forwarded to a remote server for speech processing. The grammars can then be stored locally thereon in a cache. Eventually, the grammars are provided to the recognition server 204 for use in recognition. The grammar element is used to specify grammars, either inline or referenced using an attribute.

Upon receipt of recognition results from recognition server 204 corresponding to the recognized speech, handwriting, gesture, image, etc., syntax of recognition control 290 is provided to receive the corresponding results and associate it with the corresponding field, which can include rendering of the text therein on display 34. In the illustrated embodiment, upon completion of speech recognition with the result sent back to the client, it deactivates the reco object and associates the recognized text with the corresponding field. Portions 282 and 284 operate similarly wherein unique reco objects and grammars are called for each of the fields 252 and 254 and upon receipt of the recognized text is associated with each of the fields 252 and 254. With respect to receipt of the card number field 252, the function "handle" checks the length of the card number with respect to the card type in a manner similar to that described above with respect to FIG. 7.

illustrated in FIG. 7, this code also includes a body portion 270 and a script portion 272. Also like the code illustrated in FIG. 7, the code illustrated in FIG. 8 includes indications as to the type of action to perform as well as the location of the form. Entry of information in each of the fields 250, 252 and 254 is controlled or executed by code portions 280, 282 and 284, respectively. Referring first to code portion 280, on selection of field 250, for example, by use of stylus 33 of device 30, the event "onlick" is initiated which calls or executes function "talk" in script portion 272. This action activates a grammar used for speech recognition that is associated with the type of data generally expected in field 250. This type of interaction, which involves more than one technique of input (e.g. voice and pen-click/roller) is referred as "multimodal".

It should be noted that the speech recognition extensions exemplified in Fig. 8 are not intended to have a default visual representation on the browser of the client, since for many applications it is assumed that the author will signal the speech enablement of the various components of the page by using application-specification graphical mechanisms in the source page. Nevertheless, if visual representations are desired, the extensions can so be modified.

Referring now back to the grammar, the grammar is a syntactic grammar such as but not limited to a context-free grammar, a N-grammar or a hybrid grammar. (Of course, DTMF grammars, handwriting grammars, gesture grammars and image grammars would be used when corresponding forms of recognition are employed. As used herein, a "grammar" includes information for performing recognition, and in a

(78) Generally, use of speech recognition in conjunction with architecture 200 and the client side mark-up language occurs as follows: first, the field that is associated with the speech to be given is indicated. In the illustrated embodiment, the stylus 33 is used; however, it should be understood that the present invention is not limited to use of the stylus 33 wherein any form of indication can be used such as buttons, a mouse pointer, rotatable wheels or the like. Corresponding event such as "onClick" can be provided as is well known with use of visual mark-up languages. It should be understood that the present invention is not limited to the use of the "onClick" event to indicate the start of voice, handwriting, gesture, etc commands. Any available GUI event can be used for the same purpose as well, such as "onSelect". In one embodiment, such eventing is particularly useful for it serves to indicate both the beginning and/or end of the corresponding speech. It should also be noted that the field for which the speech is directed at can be indicated by the user as well as programs running on the browser that keep track of user interactions.

At this point, it should be stated that different scenarios of speech recognition require different behaviors and/or outputs from recognition server 204. Although the starting of the recognition process is standard in all cases - an explicit start () call from uplevel browsers, or a declarative <reco> element in downlevel browsers - the means for stopping speech recognition may differ.

In the example above, a user in a multimodal application will control input into the device by, for example, tapping and holding on a pressure sensitive

display. The browser then uses a GUI event, e.g. "pen-up", to control when recognition should stop and then returns the corresponding results. However, in a voice-only scenario such as in a telephone application (discussed below) or in a hands-free application, the user has no direct control over the browser, and the recognition server 204 or the client 30, must take the responsibility of deciding when to stop recognition and return the results typically once a path through the grammar has been recognized. Further, dictation and other scenarios where intermediate results need to be returned before recognition is stopped (also known as "open microphone") not only requires an explicit stop function, but also needs to return multiple recognition results to the client 30 and/or web server 202 before the recognition process is stopped.

In one embodiment, the Reco element can include a "mode" attribute to distinguish the following three modes of recognition, which instruct the recognition server 204 how and when to return results. The return of results implies providing the "onRAcc" event or activating the "bind" elements as appropriate. In one embodiment, if the mode is unspecified, the default recognition mode can be "automatic".

FIG. 12 is a pictorial representation of operation of the "automatic" mode for speech recognition (similar modes, events, etc. can be provided for other forms of recognition). A timeline 281 indicates when the recognition server 204 is directed to begin recognition at 283, and where the recognition server 204 detects speech at 285 and determines that speech has ended at 287.

Various attributes of the Reco element control behavior of the recognition server 204. The attribute

"initialTimeout" 289 is the time between the start of recognition 283 and the detection of speech 285. If this time period is exceeded, "onSilence" event 291 will be provided from the recognition server 204, signaling that recognition has stopped. If the recognition server 204 finds the utterance to be unrecognizable, an "onNoReco" event 293 will be issued, which will also indicate that recognition has stopped.

Other attributes that can stop or cancel recognition include a "babbleTimeout" attribute 295, which is the period of time in which the recognition server 204 must return a result after detection of speech at 285. If exceeded, different events are issued according to whether an error has occurred or not. If the recognition server 204 is still processing audio, for example, in the case of an exceptionally long utterance, the "onNoReco" attribute 293 is issued. However, if the "babbleTimeout" attribute 295 is exceeded for any other reason, a recognizer error is more likely and an "onTimeout" event 297 is issued. Likewise, a "maxTimeout" attribute 299 can also be provided and is for the period of time between the start of recognition 283 and the results returned to the client 30. If this time period is exceeded, the "onTimeout" event 297 is issued.

If, however, a time period greater than an "endSilence" attribute 301 is exceeded, implying that recognition is complete, the recognition server 204 automatically stops recognition and returns its results. It should be noted that the recognition server 204 can implement a confidence measure to determine if the recognition results should be returned. If the confidence measure is below a threshold, the "cnNoReco" attribute 292 is issued, whereas if the confidence measure is above the

threshold a "onNoReco" attribute 303 and the results of recognition are issued. FIG. 12 thereby illustrates that in "automatic mode" no explicit stop () calls are made.

FIG. 13 Pictorially illustrates "single mode" operation of the recognition server 204. Attributes and events described above with respect to the "automatic mode" are applicable and are so indicated with the same reference numbers. However, in this mode of operation, a stop () call 305 is indicated on timeline 281. The stop () call 305 would correspond to an event such as "pen-up" by the user. In this mode of operation, the return of a recognition result is under the control of the explicit stop () call 305. As with all modes of operation, the "onSilence" event 291 is issued if speech is not detected within the "initialTimeout" period 289, but for this mode of operation recognition is not stopped. Similarly, a "onNoReco" event 293 generated by an unrecognizable utterance before the stop () call 305 does not stop recognition. However, if the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded recognition will stop.

FIG. 14 pictorially illustrates "multiple mode" operation of the recognition server 204. As indicated above, this mode of operation is used for an "open-microphone" or in a dictation scenario. Generally, in this mode of operation, recognition results are returned at intervals until an explicit stop () call 305 is received or the time periods associated with the "babbleTimeout" attribute 295 or the "maxTimeout" attribute 299 are exceeded. It should be noted, however, that after any "onSilence" event 291, "onReco" event 303, or "onNoReco"

event 293, which does not stop recognition, timers for the "babbleTimeout" and "maxTimeout" periods will be reset. Generally, in this mode of operation, for each phrase that is recognized, a "onReco" event 303 is issued and the result is returned until the stop () call 305 is received. If the "onsilence" event 291 is issued due to an unrecognizable utterance these events are reported but recognition will continue.

As indicated above, the associated reco object or objects for the field is activated, which includes providing at least an indication to the recognition server 204 of which grammar to use. This information can accompany the speech data recorded at the client 30 and sent to the recognition server 204. As indicated above, speech data can comprise streaming data associated with the speech entered by the user, or can include pre-processed speech data indicating speech features that are used during speech recognition. In a further embodiment, client side processing can also include normalization of the speech data such that the speech data received by the recognition server 204 is relatively consistent from client to client. This simplifies speech processing of the recognition server 204 thereby allowing easier scalability of the recognition server 204 since the recognition server can be made stateless with respect to the type of client and communication channel.

Upon receipt of the recognition result from the recognition server 204, the recognition result is associated with the corresponding field, and client-side verification or checking can be performed, if desired. Upon completion of all of the fields associated with the code currently rendered by the client, the information is sent

to web server 202 for application processing. From the foregoing, it should be clear that although the web server 202 has provided code or pages/scripts suitable for recognition to the client 30, the recognition services are not performed by the web server 202, but rather by the recognition server 204. The invention, however, does not preclude an implementation where the recognition server 204 is collocated with the web server 202, or the recognition server 204 is part of the client 30. In other words, the extensions provided herein are beneficial even when the recognition server 204 is combined with the web server 202 or client 30 because the extension provide a simple and convenient interface between these components.

While not shown in this embodiment illustrated in FIG. 8, the reco control can also include a remote audio object (RAO) to direct the appropriate speech data to the recognition server 204. The benefit for making RAO a plug-in object is to allow a different one for each different device or client because the sound interface may likely be different. In addition, the remote audio object can allow multiple reco elements to be activated at the same time. FIGS. 9A and 9B illustrate a voice-only mark-up language embodied herein as HTML with pages/scripts. As clearly illustrated, the code also includes a body portion 300 and a script portion 302. There is another extension of the markup language - prompt control 303 which include attributes like bargain. However, speech recognition is conducted differently in the voice-only embodiment of FIGS. 9A and 9B. The process is now controlled entirely by the script function "checkFilled" which will determine the unfilled fields and activate correspondent prompt and new objects. Nevertheless, grammars are activated using the objects.

same context as that described above with respect to FIG. 8, wherein speech data and the indication of the grammar to use are provided to the recognition server 204. Likewise, the output received from the recognition server 204 is associated with fields of the client (herein telephony voice browser 212).

Other features generally unique to voice-only applications is an indication to the user when speech has not been recognized. In multimodal applications such as FIG 8, 'onNoReco' simply puts null value on the displayed field to indicate no-recognition, thus no further action is required. In the voice-only embodiment, "onNoReco" 305 calls or executes a function "mumble", which forwards a word phrase to recognition server 204, that in turn, is converted to speech using a suitable text-to-speech system 307 (FIG. 5). Recognition server 204 returns an audio stream to the telephony voice browser 212, which in turn, is transmitted to phone 80 to be heard by the user. Likewise, other waveform prompts embodied in the voice-only application are also converted, when necessary, to an audio stream by recognition server 204.

It should be noted that in this example after playing the welcome prompt via function "welcome", function "checkFilled" prompts the user for each of the fields and activates the appropriate grammars, including repeating the fields that have been entered and confirming that the information is correct, which includes activation of a "confirmation" grammar. Note in this embodiment, each of the reco controls is initiated from the script portion 302, rather than the body portion of the previous example.

As another aspect of the present invention, the markup language executable on different types of client

devices (e.g. multimodal and non-display, voice-based input based client devices such as a telephone) unifies at least one of speech-related events, GUI events and telephony events for a web server interacting with each of the client devices. This is particular advantageous for it allows significant portions of the web server application to be written generically or independent of the type of client device. An example is illustrated in FIGS. 8 and 9A, 9B with the "handle" functions.

Although not shown in FIG 9, there are two more extensions to the markup language to support telephony functionality - DTMF (Dual Tone Modulated Frequency) control and call control elements or objects. DTMF works similarly to reco control. It specifies a simple grammar mapping from keypad string to text input. For example, "1" means grocery department, "2" mean pharmacy department, etc. On the other hand, call object deals with telephony functions, like call transfer and 3rd party call. The attributes, properties, methods and events are discussed in detail in the Appendix.

FIGS. 10A and 10B illustrate yet another example of a mark-up language suitable for a voice-only mode of operation. In this embodiment, the user is allowed to have some control over when information is entered or spoken. In other words, although the system may initiate or otherwise direct the user to begin speaking, the user may offer more information than what was initially asked for. This is an example of "mixed initiative". Generally, in this form of dialog interaction, the user is permitted to share the dialog initiative with the system. Besides the example indicated above and discussed below in detail where the user provides more information then requested by a prompt,

the user could also switch tasks when not prompted to do so.

In the example of FIGS 10A and 10B, a grammar identified as "do_field" includes the information associated with the grammars "g_card_types", "g_card_num" and "g_expiry_date". In this example, telephony voice browser 212 sends speech data received from phone 80 and an indication to use the "do_field" grammar to recognition server 204 upon receipt of the recognized speech as denoted by "onReco", the function "handle" is called or executed that includes associating the values for any or all of the fields recognized from the speech data. In other words, the result obtained from the recognition server 204 also includes indications for each of the fields. This information is parsed and associated with the corresponding fields according to binding rules specified in 405. As indicated in FIG. 5, the recognition server 204 can include a parser 309.

From FIGS. 7, 8, 9A, 9B, 10A and 10B, a very similar web development framework is used. Data presentation is also very similar in each of these cases. In addition, the separation of data presentation and flow controls allow maximum reusability between different applications (system initiative and mixed-initiative), or different modalities (GUI web-based, voice-only and multimodal). This also allows a natural extension from voice-only operation through a telephone to a multimodal operation when phones include displays and functionalities similar to device 30. Appendix A provides further details of the controls and objects discussed above.

As indicated above, uplevel browsers can use

scripting in order to perform various needs such as

invoking the function "handle" in the above examples to assign the recognition results. In the embodiments described above and as further described in Appendix A at 2.1.2, the "bind" element will parse the recognition results and assign values wherein the "bind" element is a subelement or child element of the "reco" element. Although scripting can be useful, it is believed by many not to be always the best form of browser implementation due, for example, to security concerns. Therefore, in yet another embodiment or aspect of the present invention, the "bind" element is a high level element (similar to "reco") and is provided with other richer properties, which can in effect mimic scripting without scripting per se.

Without using scripting or without using the aspect of the present invention discussed below, some of the capabilities discussed below such as sophisticated dialog effects could only be accomplished by submitting a page back to the web server 202, executing the application logic thereon to generate a new page, and sending the page back to the client device. This aspect of the present invention allows a programmer to invoke methods on objects of the page without incurring a server roundtrip. In the embodiments described above, the "bind" element has only the attributes "targetElement" and "targetAttribute" for assigning recognition results to a field in the form or web page. In a further embodiment, the "bind" element also includes a "TargetMethod", which is added for object method invocation. The use and capability of "TargetMethod" is the principal technique for mimicking scripting. For example, the following syntax can be used to invoke the "X" method of the object "OBJ":

```
(88) <bind TargetElement = "Obj1" TargetMethod =
"X" ...>
```

Note that although the examples shown here follow the HTML/XHTML event syntax, those skilled at art should find it straightforward to generalize the <bind> usage to other eventing mechanisms, including but not limited to W3C Document Object Model Level 2 or Level 3 eventing standard, ECMAS Common language Infrastructure (CLI) event model, Java Programming language event mode-, W3C Synchronous Multimedia Integration Language (SMIL), and the upcoming W3C XML Events standard proposal.

FIGS. 15A and 15B are a page of mark-up language executable on a client, particularly a downlevel browser. In this example, the user is asked through audio prompts for a drink. The system then confirms what drink was ordered. Depending on the recognition results, the "bind" element guides the execution using declared logic. When the drink is confirmed, the form is submitted back to the web server 202, all without scripting.

Generally, the mark-up example of FIGS. 15A and 15B includes a data section 350, a speech section 352 and user interface sections 354, 356 and 358. Section 354 receives the recognition result from the general inquiry as to what drink the user would like and directs interactive recognition flow to either reprompting, inquiring as to whether cream or sugar is needed, or confirming the drink ordered. In particular, section 356 receives a recognition result when cream or sugar is also ordered. Section 358 receives the recognition result for confirmation of the drink. Section 360 is a call control section employing a new messaging object "SMEX", which is discussed further below.

"X" ...>.

As indicated above, the "bind" element of this aspect of the present invention includes object method invocation, which will initiate user interaction in the example of FIGS. 15A and 15B by playing the "welcome" prompt when the "start" method on the "welcome" object is executed at 361.

The user is then asked "Do you want coke, coffee or orange juice?" by executing the "start" method of the "asked" object at 362. Recognition is then performed by invoking the "start" method on the recognition "reco_drink" object at 363.

The mark-up of section 354 is then executed wherein the grammar used by the recognition server 204 is provided by the XPath statement "/drink types". Note that although this example utilizes W3C XPath language, those skilled at art should find it straightforward to extend the concept to other standard languages, including but not limited to W3C XML Query Language (XQL). As specified by the "bind" element 364, if the recognition result received from the recognition server 204 has a confidence score less than 10, the prompt object "reprompt" is executed at 366, followed by the prompt object "ask" 368, at which point, the recognition object "reco_drink" is reinitialized at 370. If the recognition result returned is "coffee" having a confidence greater than 10, the field "drink" is assigned the value of the recognition result at 372 and the user is then prompted as to whether or not he/she would like cream or sugar by the prompt object "cream_sugar" at 374. The recognition object "reco_cream sugar" in section 356 is then invoked at 376. Otherwise, if the confidence score of the recognition result is greater than 10, but not coffee, the field drink is then again assigned at 378. Confirmation

means when the "onNORECO" event is sent, the method "start" of the object "confirm", followed by invocation of the prompt object "reco.yesno" in section 358 at 382. If the user answers "yes" with a confidence score greater than 10, the prompt object "thanks" is played at 384 and then the form is then submitted at 386. Otherwise, if the user answers "no" or the confidence score of the recognition result is less than 10, the prompt object "retry" is executed at 390, once again followed by the prompt object "ask" being executed at 392, and invocation of the "reco_drink" recognition object at 394.

From the foregoing example, the "bind" element allows multiple invocations of methods as indicated in sections 354, 356 or 359. If desired, multiple assignments of the recognized result can also be declared. In the embodiment illustrated, when multiple assignments and method invocations are declared, they are executed in the document order.

In a further embodiment, a convention for passing method arguments is also provided. In other words, some methods may require a list of arguments. This is achieved by using "arg" subelement. For example, given the following mark-up:

```
<bind TargetElement = "OBJ" TargetMethod = "F"><arg>X</arg><arg>Y</arg></bind>
```

is equivalent to "OBJ.F(X,Y)", or "OBJ" is an object that has a method "F" with parameters or arguments "X" and "Y".

The "bind" element can also include an "event" attribute, which declares the bind element is intended for which event. For example, the mark-up:

```
<bind event = "onNORECO" = TargetElement = "prompt" TargetMethod = "start"/>
```

means when the "onNORECO" event is sent, the method "start" of the object "prompt" will be invoked. To be consistent with use of the "bind" element as a child element of the "Reco" element as described above for example with respect to FIG. 8, the default attribute for the "bind" element is "onReco".

The "bind" element as a high level element can include any of the events specified in section 2.4 of the Appendix. In addition, the "bind" element can also include a "onError" event having a "status" attribute that can be accessed and used to direct program flow. To the extent other events of the "bind" element have "status" attributes, these can be accessed as well.

In addition to checking the conditions of the recognition result, the current document or page being executed can also be checked as well. In particular, both the "test" and the "value" attribute can be extended to include a "host" primitive that refers to the root node of the containing document. For example, referring back to FIGS. 15A, 15D, the example contained therein has additional logic at section 354 to ask whether the user wants cream or sugar when he/she asks for coffee. The flags for adding cream or sugar and thus invocation of section 356, will be turned on only if the drink field is "coffee" as specified by the markup

```
"hcst() /get_drink/drink='coffee'.
```

It should also be noted, the "bind" element is not only applicable to recognition results from the speech server 204 and receive or assign values in the document, but also to message objects (herein denoted by "smex", for example, from applications running on the client device. In the example of FIGS. 15A and 15B, the page is executed when

a telephony application running on the client device detects a call. In section 360, the "bind" element executes or plays the "welcome" prompt and begins recognition by executing the "reco_drink" object, when the message received "/Call_connected" is received. Like the recognition results received from the speech server 204, the messages received can vary greatly. Some of the messages are well-defined in order to initiate desired program flow. Others can be received and processed (e.g. parsed just like received recognition results of the recognition server. For example, this allows the markup to be used like a natural language parser of text from a keyboard. The reco element in Appendix A includes a property for performing this function. Likewise, the prompt element can be used to provide a textual message for dynamic content or audio wave files by using the property "innertext", also further explained in the Appendix A. Eventing can be similar to eventing for recognition results. For example, eventing can include "onReceived", which gets sent when the message source (e.g. application running on the client device) has a message available for the browser.

The "smex" or message object thus allows the markup tags discussed herein to be extended to other components or applications running on the client device. As another example, the message object can be used to communicate to a TTY component for the hearing impaired running on the client device. Rather than using speech recognition, the TTY component will provide a message of what the user has typed. This message is then used as if a recognition result was received from the recognition server wherein the message can be parsed and assigned to fields of the form, or other processing can take place using the

```
"reco", "grammar" or "bind" elements described above.  
Further discussion of the message or "smex" object is  
provided in Appendix A.  
The "bind" element can also include a "for"  
attribute, which allows its action to be attached to other  
objects on the page. For example, a mark-up such as:  
<bind for = "prompt 1" event = "onComplete"  
targetElement = "prompt 2" = targetMethod =  
"start" />.
```

will invoke the start method of the object "prompt 2" when the object "prompt 1" sends the event "onComplete". Referring back to FIG. 5, web server 202 can include a server side plug-in declarative authoring tool or module 320 (e.g. ASP or ASP+ by Microsoft Corporation, JSP, or the like). Server side plug-in module 320 can dynamically generate client-side mark-ups and even a specific form of mark-up for the type of client accessing the web server 202. The client information can be provided to the web server 202 upon initial establishment of the client/server relationship, or the web server 202 can include modules or routines to detect the capabilities of the client. In this manner, server side plug-in module 320 can generate a client side mark-up for each of the voice recognition scenarios, i.e. voice only through phone 80 or multimoda: for device 30. By using a consistent client side model (reco and prompt controls that can be used in each application), application authoring for many different clients is significantly easier.

In addition to dynamically generating client side mark-ups, high-level dialog modules, like getting credit card information illustrated in FIG. 6 with a mark-up examples of FIGS. 8, 9A and 9B, can be implemented as a

server-side control as stored in store 324 for use by developers in application authoring. In general, the high-level dialog modules 324 would generate dynamically client-side markup and script in both voice-only and multimodal scenarios based on parameters specified by developers. The high-level dialog modules can include parameters to generate client-side mark-ups to fit the developers' needs. For example, a credit card information module can include a parameter indicating what types of credit cards the client-side mark-up script should allow. A sample ASP+ page using in server side plug-in module 320 is illustrated in FIG. 11.

Although the present invention has been described with reference to preferred embodiments, workers skilled in the art will recognize that changes may be made in form and detail without departing from the spirit and scope of the invention.

APPENDIX A

1 Introduction

The following tags are a set of markup elements that allows a document to use speech as an input or output medium. The tags are designed to be self-contained XML that can be imbedded into any SGML derived markup languages such as HTML, XHTML, CHTML, SMIL, WML and the like. The tags used herein are similar to SAPI 5.0, which are known methods available from Microsoft Corporation of Redmond, Washington. The tags, elements, attributes, properties, return values, etc. are merely exemplary and should not be considered limiting. Although exemplified herein for speech and DTMF recognition, similar tags can be provided for other forms of recognition.

The main elements herein discussed are:

```
<prompt ...> for speech synthesis configuration and
prompt playing
<reco ...> for recognizer configuration and recognition
execution and post-processing
<grammar ...> for specifying input grammar resources
<bind ...> for processing of recognition results
<dtmf ...> for configuration and control of DTMF
```

2 Reco

The Reco element is used to specify possible user inputs and a means for dealing with the input results. As such, its main elements can be <grammar> and <bind>, and it contains resources for configuring recognizer properties.

Reco elements are activated programmatically in uplevel browsers via Start and Stop methods, or in SMI-enabled browsers by using SMI commands. They are considered active declaratively in downlevel browsers (i.e. non script-supporting browsers) by their presence on the page. In order to permit the activation of multiple grammars in parallel, multiple Reco elements may be considered active simultaneously.

Recos may also take a particular mode - 'automatic', 'single' or 'multiple' - to distinguish the kind of recognition scenarios which they enable and the behavior of the recognition platform.

2.1 Reco content

The Reco element contains one or more grammars and optionally a set of bind elements which inspect the results of recognition and copy the relevant portions to values in the containing page.

In uplevel browsers, Reco supports the programmatic activation and deactivation of individual grammar rules. Note also that all top-level rules in a grammar are active by default for a recognition context.

2.1.1 <grammar> element

The grammar element is used to specify grammars, either inline or referenced using the src attribute. At least one grammar (either inline or referenced) is typically specified. Inline grammars can be text-based grammar formats, while referenced grammars can be text-based or binary type. Multiple grammar elements may be specified. If

more than one grammar element is specified, the rules within grammars are added as extra rules within the same grammar. Any rules with the same name will be overwritten.

Attributes:

- src: Optional if inline grammar is specified. URI of the grammar to be included. Note that all top-level rules in a grammar are active by default for a recognition context.
- langID: Optional. String indicating which language speech engine should use. The string format follows the XML:lang definition. For example, langID="en-us" denotes US English. This attribute is only effective when the langID is not specified in the grammar URI. -f unspecified, defaults to US English.

If the langID is specified in multiple places then langID follows a precedence order from the lowest scope - remote grammar file (i.e. language id is specified within the grammar file) followed by grammar element followed by reco element.

```
<grammar src="FromCity.xml" />
or
<grammar>
  <rule topLevel="active">
    <p>from </p>
    <ruleRef name="cities" />
  </rule>
  <rule name="cities">
    <l>
      <p> Cambridge </p>
      <p> Seattle </p>
      <p> London </p>
    </l>
  </rule>;
</grammar>
```

</grammar>

<grammar>

If both a src-referenced grammar and an inline grammar are specified, the inline rules are added to the referenced rules, and any rules with the same name will be overwritten.

2.1.2 <bind> element

The bind element is used to bind values from the recognition results into the page.

The recognition results consumed by the bind element can be an XML document containing a semantic markup language (SML) for specifying recognition results. Its contents include semantic values, actual words spoken, and confidence scores. SML could also include alternate recognition choices (as in an N-best recognition result). A sample SML document for the utterance "I'd like to travel from Seattle to Boston" is illustrated below:

```
<sml confidence="40">
  <travel text="I'd like to travel from Seattle to
  Boston">
    <origin_city confidence="45"> Seattle
  </origin_city>
    <dest_city confidence="35"> Boston
  </dest_city>
  </travel>
</sml>
```

Since an inline-grammar recognition is assumed to produce an XML document – in semantic markup language, or SML – the values to be bound from the SML document are referenced using an XPath query. And since the elements in the page into which the values will be bound should be uniquely identified

(they are likely to be form controls), these target elements are referenced directly.

Attributes:

- **targetElement**: Required. The element to which the value content from the SML will be assigned (as in W3C SML 2.0).
- **targetAttribute**: Optional. The attribute of the target element to which the value content from the SML will be assigned (as with the attributeName attribute in SML 2.0). If unspecified, defaults to "value".
- **test**: Optional. An XSLT Pattern (as in the W3C XML DOM specification) string indicating the condition under which the recognition result will be assigned. Default condition is true.
- **value**: Required. An XML (as in the W3C XML DOM specification) string that specifies the value from the recognition result document to be assigned to the target e-element.

Example:

So given the above SML return, the following reco element uses bind to transfer the values in origin_city and dest_city into the target page elements txtBoxOrigin and txtBoxDest:

```
<input name="txtBoxOrigin" type="text" />
<input name="txtBoxDest" type="text" />
<reco id="travel">
  <grammar src=".//city.xml" />
```

(100)

```

<bind      targetElement="txtBoxOrigin"
    value="//origin_city" />
<bind      targetElement="txtBoxDest"
    value="//dest_city" />
</reco>
```

This binding may be conditional, as in the following example, where a test is made on the confidence attribute of the dest_city result as a pre-condition to the bind operation:

```

<bind targetElement="txtBoxDest"
    value="//dest_city"
    test="/sml/dest_city(@confidence >= 40)"
/>
```

The bind element is a simple declarative means of processing recognition results on down-level or up-level browsers. For more complex processing, the reco DOM object supported by up-level browsers implements the onReco event handler to permit programmatic script analysis and post-processing of the recognition return.

2.2 Attributes and properties

The following attributes are supported by all browsers, and the properties by up-level browsers.

2.2.1 Attributes

The following attributes of Reco are used to configure the speech recognizer for a dialog turn.

- **initialTimeout:** Optional. The time in milliseconds between start of recognition and the detection of speech. This value is passed to the recognition platform, and if exceeded, an onSilence event will be provided from the recognition platform (see 2.4.2). If not specified, the speech Platform will use a default value.
- **babbleTimeout:** Optional. The period of time in milliseconds in which the recognizer must return a result after detection of speech. For recos in automatic and single mode, this applies to the period between speech detection and the stop call. For recos in 'multiple' mode, this timeout applies to the period between speech detection and each recognition return - i.e. the period is restarted after each return of results or other event. If exceeded, different events are thrown according to whether an error has occurred or not. If the recognizer is still processing audio - e.g. in the case of an exceptionally long utterance - the onNoReco event is thrown, with status code 13 (see 2.4.4). If the timeout is exceeded for any other reason, however, a recognizer error is more likely, and the onTimout event is thrown. If not specified, the speech Platform will default to an internal value.
- **maxTimeout:** Optional. The period of time in milliseconds between recognition start and results returned to the browser. If exceeded, the onTimeout event is thrown by the browser - this caters for network or recognizer failure in distributed

environments. For recos in 'multiple' mode, as with `babbleTimeout`, the period is restarted after the return of each recognition or other event. Note that the `maxLength` attribute should be greater than or equal to the sum of `initialTimeout` and `babbleTimeout`. If not specified, the value will be a browser default.

- endsilence:** Optional. For Recos in automatic mode, the period of silence in milliseconds after the end of an utterance which must be free of speech after which the recognition results are returned. Ignored for recos of modes other than automatic. If unspecified, defaults to platform internal value.
- reject:** Optional. The recognition rejection threshold, below which the platform will throw the 'no reco' event. If not specified, the speech platform will use a default value. Confidence scores range between 0 and 100 (integer). Reject values lie in between.
- server:** Optional. URI of speech platform (for use when the tag interpreter and recognition platform are not co-located). An example value might be `server=protocol://yourspeechplatform`. An application writer is also able to provide speech platform specific settings by adding a querystring to the URI string, eg `protocol://yourspeechplatform?bargeInEnergyThreshold=0.5`.

- langID:** Optional. String indicating which language speech engine should use. The string format follows the `xml:lang` definition. For example, `langID="en-us"`

denotes US English. This attribute is only effective when the `langID` is not specified in the grammar element (see 2.1.1).

- mode:** Optional. String specifying the recognition mode to be followed. If unspecified, defaults to "automatic" mode.

2.2.2 Properties

- The following properties contain the results returned by the recognition process (these are supported by up-level browsers).
 - recоБResult** Read-only. The results of recognition, held in an XML DOM node object containing semantic markup in language (SML), as described in 2.1.2. In case of no recognition, the property returns null.
 - text** Read/write. A string holding the text of the words recognized (i.e., a shorthand for contents of the text attribute of the highest level element in the SML recognition return in `recоБResult` in a read mode. In write mode, a string can be assigned, which will then be parsed as if the string corresponds to the recognition result. The write mode allows extension of the markup tags and processing thereof to other components or applications on the client device. The string can be obtained from the "smex" message object.
 - status:** Read-only. Status code returned by the recognition platform. Possible values are 0 for successful recognition, or the failure values -1 to -4 (as defined in the exceptions possible on the Start method (section 2.3.1) and `Activate` method (section

2.3.4), and statuses -11 to -15 set on the reception of recognizer events (see 2.4).

2.3 Object methods

Reco activation and grammar activation may be controlled using the following methods in the Reco's DOM object. With these methods, uplevel browsers can start and stop Reco objects, cancel recognitions in progress, and activate and deactivate individual grammar top-level rules (uplevel browsers only).

2.3.1 Start

The Start method starts the recognition process, using as active grammars all the top-level rules for the recognition context which have not been explicitly deactivated.

Syntax:

```
Object.Start( )
```

Return value:

None.

Exception:

None.

2.3.2 Stop

The Stop method is a call to end the recognition process. The Reco object stops recording audio, and the recognizer

returns recognition results on the audio received up to the point where recording was stopped. All the recognition resources used by Reco are released, and its grammars deactivated. (Note that this method need not be used explicitly for typical recognitions in automatic mode, since the recognizer itself will stop the reco object of endpoint detection after recognizing a complete sentence.) If the Reco has not been started, the call has no effect.

Syntax:

```
Object.Stop( )
```

Return value:

None.

Exception:

None.

2.3.3 Cancel

The Cancel method stops the audio feed to the recognizer, deactivates the grammar and releases the recognizer and discards any recognition results. The browser will disregard a recognition result for canceled recognition. If the recognizer has not been started, the call has no effect.

Syntax:

```
Object.Cancel( )
```

Return value:

None.

Exception:

None.

Syntax:

```
Object.Cancel( )
```

Return value:

None.

Exception:

None.

2.3.2 Stop

The Stop method is a call to end the recognition process. The Reco object stops recording audio, and the recognizer

returns recognition results on the audio received up to the point where recording was stopped. All the recognition resources used by Reco are released, and its grammars deactivated. (Note that this method need not be used explicitly for typical recognitions in automatic mode, since the recognizer itself will stop the reco object of endpoint detection after recognizing a complete sentence.) If the Reco has not been started, the call has no effect.

2.3 Object methods

Reco activation and grammar activation may be controlled using the following methods in the Reco's DOM object. With these methods, uplevel browsers can start and stop Reco objects, cancel recognitions in progress, and activate and deactivate individual grammar top-level rules (uplevel browsers only).

Syntax:

```
Object.Stop( )
```

Return value:

None.

Exception:

None.

Syntax:

```
Object.Cancel( )
```

Return value:

None.

Exception:

None.

Syntax:

```
Object.Cancel( )
```

Return value:

None.

Exception:

None.

2.3.4 Activate

The `Activate` method activates a top-level rule in the context free grammar (CFG). Activation must be called before recognition begins, since it will have no effect during a 'started' recognition process. Note that all the grammar top-level rules for the recognition context which have not been explicitly deactivated are already treated as active.

Syntax:

```
Object.Activate(strName);
```

Parameters:

- o `strName`: Required. Rule name to be activated.

Return value:

None.

Exception:

None.

2.3.5 Deactivate

The `method deactivate` a top-level rule in the grammar. If the rule does not exist, the method has no effect.

Syntax:

```
Object.Deactivate(strName);
```

Parameters:

- o `strName`: Required. Rule name to be deactivated. An empty string deactivates all rules.

Return value:

None.

Exception:

None.

2.4 Reco events

The Reco DOM object supports the following events, whose handlers may be specified as attributes of the `<reco` element.

2.4.1 onReco:

This event gets fired when the recognizer has a recognition result available for the browser. For `reco` in automatic mode, this event stops the recognition process automatically and clears resources (see 2.3.2). `OnReco` is typically used for programmatic analysis of the recognition result and processing of the result into the page.

Syntax:

```
<reco onReco="handler" >
```

```
Object.onReco = handler;
```

```
Object.onReco =
```

```
GetRef("handler");
```

Event Object Info:

Bubbles	No
To invoke	User says something
Default	Return recognition result object action

Event Properties:

Although the event `handler` does not receive properties directly, the `handler` can query the event object for data (see the use of the `event` object in the example below).

Example
 The following XHTML fragment uses onReco to call a script to parse the recognition outcome and assign the values to the proper fields.

```
<input name="txtBoxOrigin" type="text" />
<input name="txtBoxDest" type="text" />
<reco onReco="processCityRecognition()"/>
<grammar src="/grammars/cities.xml" />
```

```
</reco>
```

```
<script><![CDATA[
function processCityRecognition () {
  smlResult = event.srcElement.recoResult;

  origNode =
    smlResult.selectSingleNode("//origin_city");
  if (origNode != null) txtBoxOrigin.value =
    origNode.text;

  destNode =
    smlResult.selectSingleNode("//desc_city");
  if (destNode != null) txtBoxDest.value =
    destNode.text;
}
```

This event cancels the recognition process automatically for the automatic recognition mode.

Syntax:

Inline HTML	<reco onSilence="handler" ...>
Event Property (in ECMAScript)	Object.onSilence = handler GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Recognize did not detect speech within the period specified in the initialTimeout attribute.
Default	Set status = -1; action

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.3 onTimeout

onTimeout handles two types of event which typically reflect errors from the speech platform.

- It handles the event thrown by the tags interpreter which signals that the period specified in the maxTime attribute (see 2.2.1) expired before recognition was completed. This event will typically reflect problems that could occur in a distributed architecture.

2.4.2 onSilence:

onSilence handles the event of no speech detected by the recognition platform before the duration of time specified in the initialTimeout attribute on the Reco (see 2.2.1).

- It also handles (ii) the event thrown by the speech recognition platform when recognition has begun but processing has stopped without a recognition within the period specified by babbleTimeout (see 2.2.1).

This event cancels the recognition process automatically.

Syntax:

Inline HTML:	<Reco onTimeout="handler">
Event property (in ECMAScript):	Object.onTimeout = handler; Object.setTimeout = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Thrown by the browser when the period set by the maxTime attribute expires before recognition is stopped.
Default action	Set reco status to -12.

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

2.4.4 onNoReco:

onNoReco is a handler for the event thrown by the speech recognition platform when it is unable to return valid recognition results. The different cases in which this may

- happen are distinguished by status code. The event stops the recognition process automatically.

Syntax:

Inline HTML:	<Reco onNoReco ="handler" >
Event property	Object.onNoReco = handler; Object.onNoReco = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	Recognizer detects sound but is unable to interpret the utterance.
Default action	Set status Property and return null
	recognition result. Status codes are set as follows:

status -13: sound was detected but no speech was able to be interpreted;
status -14: some speech was detected and interpreted but rejected with insufficient confidence (for threshold setting, see the reject attribute in 2.2.1).
status -15: speech was detected and interpreted, but a complete recognition was unable to be returned between the detection of speech and the duration specified in the babbleTimeout attribute (see 2.2.1).

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3 Prompt

The prompt element is used to specify system output. Its content may be one or more of the following:

- inline or referenced text, which may be marked up with prosodic or other speech output information;
- variable values retrieved at render time from the containing document;
- links to audio files.

Prompt elements may be interpreted declaratively by downlevel browsers (or activated by SMIL commands), or by object methods on uplevel browsers.

3.1 Prompt content

The prompt element contains the resources for system output, either as text or references to audio files, or both.

Simple prompts need specify only the text required for output, e.g.:

```
<prompt id="Welcome">
    Thank you for calling ACME weather report.
</prompt>
```

This simple text may also contain further markup of any of the kinds described below.

3.1.1 Speech Synthesis markup

Any format of speech synthesis markup language can be used inside the prompt element. (This format may be specified in the 'tts' attribute described in 3.2.1.) The following example shows text with an instruction to emphasize certain words within it:

```
<prompt id="giveBalance">
    You have <emph> five dollars </emph> left in your account.
</prompt>
```

3.1.2 Dynamic content

The actual content of the prompt may need to be computed on the client just before the prompt is output. In order to confirm a particular value, for example, the value needs to be dereferenced in a variable. The value element may be used for this purpose.

Value Element

value: Optional. Retrieves the values of an element in the document.

Attributes:

- **targetElement:** Optional. Either href or targetElement must be specified. The id of the element containing the value to be retrieved.
- **targetAttribute:** Optional. The attribute of the element from which the value will be retrieved.

- **href**: Optional. The URI of an audio segment. **href** will override **targetElement** if both are present.

- **href:** Optional. The URI of an audio segment. href will override targetElement if both are present.
 - The targetElement attribute is used to reference an element within the containing document. The content of the element whose id is specified by targetElement is inserted into the text to be synthesized. If the desired content is held in attribute of the element, the targetAttribute attribute will be used to specify the necessary attribute on the targetElement. This is useful for dereferencing the values of HTML form controls, for example. In the following illustration, the "value" attributes of the "txtBoxOrigin" and "txtBoxDest" elements are inserted into the text before a prompt is output

After the beep, please record your message.
<value href="/wav/beep.wav" />

After the beep, please record your message.
<value href="/wav/beep.wav" />

The `targetElement` attribute is used to reference an element within the containing document. The content of the element whose id is specified by `targetElement` is inserted into the text to be synthesized. If the desired content is held in an attribute of the element, the `targetAttribute` attribute may be used to specify the necessary attribute on the `targetElement`. This is useful for dereferencing the values in HTML form controls, for example. In the following illustration, the "value" attributes of the "txtBoxOrigin" and "txtBoxDest" elements are inserted into the text before the prompt is output.

3.1.4 Referenced prompts

Instead of specifying content inline, the `src` attribute may be used with an empty element to reference external content via URI, as in:

```
<prompt id="Welcome" src="~/ACMEWeatherPrompts#Welcome"/>
```

The target of the `src` attribute can hold any or all of the above content specified for inline prompts.

3.2 Attributes and properties

The prompt element holds the following attributes (downlevel browsers) and properties (downlevel and uplevel browsers).

3.2.1 Attributes

- tts**: Optional. The markup language type for text-to-speech synthesis. Default is "SAPI 5".
- src**: Optional if an inline prompt is specified. The URL of a referenced prompt (see §.1.4).
- bargain**: Optional. Integer. The period of time in milliseconds from start of prompt to when playback can be interrupted by the human listener. Default is infinite, i.e., no bargain is allowed. Bargain=0 allows immediate bargain. This applies to whichever kind of bargain is supported by platform. Either keyword or energy-based bargain times can be

3.2.2 tC

```
<value targetElement="txtBoxDest"
      targetAttribute="value" />
?
</prompt>
```

3.2.3 Audio files

The value element may also be used to refer to a pre-recorded audio file for playing instead of, or within, a synthesized prompt. The following example plays a beep at the end of the prompt:

```
<prompt id="Confirm">
  Do you want to travel from
  <value targetElement="txtBoxOrigin"
        targetAttribute="value" />
  <audio src="beep.mp3" />
</prompt>
```

(116) 特關2003-131772

configured in this way, depending on which is enabled at the time the reco is started.

- **prefetch:** Optional. A Boolean flag indicating whether the prompt should be immediately synthesized and cached at browser when the page is loaded. Default is false.

3.2.2 Properties
Uplevel browsers support the following properties in the prompt's DOM object.

- **bookmark:** Read-only. A string object recording the text of the last synthesis bookmark encountered.
- **status:** Read-only. Status code returned by the speech platform.
- **Innertext:** Read-only. This property will provide the text transcription of the prompt, which would be sent to the synthesizer. For instance, if a prompt comprises playing an audio wave file, this property provides a text version of that prompt (often stored with the audio wave file), which can then be displayed, or otherwise used, for example, by providing the text version of the prompt to a component or application running on the client device. The innertext property can also be used to provide text versions of prompts containing dynamic content.

(117)

特關2003-131772

in progress, and change the speed and volume of the synthesized speech.

- **Start:**
Start playback of the prompt. Unless an argument is given, the method plays the contents of the object. Only a single prompt object is considered 'started' at a given time, so if Start is called in succession, all playbacks are played in sequence.

Syntax:

```
Object.Start([strText]);
```

Parameters:

- **strText:** the text to be sent to the synthesizer. If present, this argument overrides the contents of the object.

Return value:

None.

Exception:

- Set status = -1 and fires an onComplete event if the audio buffer is already released by the server.

3.3.2 Pause:

- Pause playback without flushing the audio buffer. This method has no effect if playback is paused or stopped.

Syntax:

Object.Pause();

Return value:

None.

Exception:

None.

3.3 Prompt methods

Prompt playing may be controlled using the following methods in the prompt's DOM object. In this way, uplevel browsers can start and stop prompt objects, pause and resume prompts

(118) 特殊2003-131772

3.3.3 Resume

Resume playback without flushing the audio buffer. This method has no effect if playback has not been paused.

Syntax:

Object.Resume();

Return value:

None.

Exception:

Throws an exception when resume fails.

3.3.4 Stop

Stop playback, if not already, and flush the audio buffer. If the playback has already been stopped, the method simply flushes the audio buffer.

Syntax:

Object.Stop();

Return value:

None.

Exception:

None.

3.3.5 Change

Change speed and/or volume of playback. Change may be called during playback.

Syntax:

Object.Change(speed, volume);

Parameters:

- o speed: Required. The factor to change. Speed=2.0 means double the current rate, speed=0.5 means

(119) 特殊2003-131772

halve the current rate, speed=0 means to restore the default value.

c volume: Required. The factor to change. Volume=2.0 means double the current volume, volume =0.5 means halve the current volume, volume =0 means to restore the default value.

Return value:

None.

Exception:

None.

3.3.6 Prompt control example

The following example shows how control of the prompt using the methods above might be authored for a platform which does not support a keyword barge-in mechanism.

```
<html>
<t:2>Prompt control</t:2>
<head>
<script>
<!--
function checkKBargein() {
    news.change(-0.5); // turn down the volume
    while verifying
        if (keyword.text == "") // result is below
            threshold
                news.change(1.0, 2.0); // restore the volume
                keyword.start(); // restart the recognition
            else
                news.stop(); // keyword detected! Stop the
                prompt
                // Do whatever that is necessary
}
-->
</script>
<script for="window" event="onload">
<!--
```

```
// Object.Change(speed, volume);
// Parameters:
// o speed: Required. The factor to change. Speed=2.0
means double the current rate, speed=0.5 means
```

```

news.Start(); keyword.Start();
//  

</script>
</head>
<body id="news" bargein="0">
<!--<prompt id="news" bargein="0">
Stocks turned in another lackluster performance Wednesday  

as investors received little incentive to make any big moves  

ahead of next week's Federal Reserve meeting. The tech-heavy  

Nasdaq Composite Index dropped 42.51 points to close at  

2156.26. The Dow Jones Industrial Average fell 17.05 points  

to 10866.46 after an early-afternoon rally failed.  

-->
<!--  

</prompt>  

<reco id="keyword"  

reject="70"  

onRecog="checkKWBargein()">  

<grammar src="http://denali/news_bargein_grammar.xml" />  

</reco>
</body>
</html>

```

3.4 Prompt events

The prompt DOM object supports the following events, whose handlers may be specified as attributes of the prompt element..

3.4.1 onBookmark

Fires when a synthesis bookmark is encountered. The event does not pause the playback.

Syntax:

Inline HTML	<prompt onBookmark="handler" ...>
Event property	Object.onBookmark = handler

Syntax:

Inline HTML	<prompt onBargein="handler" ...>
Event property	Object.onBargein = handler

Event Object Info:

Bubbles	No
To invoke	A bargein event is encountered

Event Object Info:

Bubbles	No
To invoke	A bookmark in the rendered string is encountered
Default action	Returns the bookmark string

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.2 onBargein:
Fires when a user's barge-in event is detected. (Null)
that determining what constitutes a barge-in event, as
energy detection or keyword recognition, is up to the
platform.) A specification of this event handler does
not automatically turn the barge-in on.

Default	None
action	

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.3 onComplete:

Fires when the prompt playback reaches the end or exceptions (as defined above) are encountered.

Syntax:

Inline HTML	<prompt onComplete="handler" ...>
Event property	Object. onComplete = handler Object. onComplete = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	A prompt playback completes
Default	Set status = 0 if playback completes normally, otherwise set status as specified above.
action	

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

3.4.4 Using bookmarks and events

The following example shows how bookmark events can be used to determine the semantics of a user response - either a correction to a departure city or the provision of a destination city - in terms of when bargeIn happened during the prompt output. The onBargeIn handler calls a script which sets a global 'mark' variable to the last bookmark encountered in the prompt, and the value of this 'mark' is used in the reco's postprocessing function ('heard') to set the correct value.

```
<script><![CDATA[
    var mark;
    function interrupt( ) {
        mark = event.srcElement.bookmark;
    }
    function ProcessCityConfirm( ) {
        confirm.stop(); // flush the audio buffer
        if (mark == "mark_origin_city") {
            txtBoxOrigin.value =
                event.srcElement.text;
        } else
            txtBoxDest.value =
                event.srcElement.text;
    }
})</script>
<body>
<input name="txtBoxOrigin" value="Seattle"
type="text"/>
<input name="txtBoxDest" type="text" />
...
<prompt id="confirm" onBargeIn="interrupt()" bargain="0">
From <bookmark mark="mark_origin_city" targetAttribute="value" />
Please say <bookmark mark="mark_dest_city" /> the
destination city you want to travel to.
</prompt>
<reco onReco="ProcessCityConfirm()" >
<grammar src="/gram/103/cities.xml" />
</reco>
```

(124) 特開2003-131772

特開2003-131772

...</body>

4 DINE

Creates a DTMF recognition object. The object can be instantiated using inline markup language syntax or in scripting. When activated, DTMF can cause prompt object to fire a barge-in event. It should be noted the tags and eventing discussed below with respect to DTMF recognition and call control discussed in Section 5 generally pertain to interaction between the voice browser 216 and media server 224.

4.1 Content

- dmfgrammar: for inline grammar.
 - bind: assign DMF conversion result to proper field.

Attributes:

 - targetElement: Required. The element to which a partial recognition result will be assigned to (cf. same as in W3C SHIL 2.0).
 - targetAttribute: the attribute of the target element to which the recognition result will be assigned to (cf. same as in SHIL 2.0). Default is "valu".
 - test: condition for the assignment. Default is true.

Digitized by srujanika@gmail.com

- **targetElement:** Required. The element to which a partial recognition result will be assigned to (cf. same as in W3C [SMIL 2.0](#)).
 - **Example 3:** How to allow both speech and DTMF inputs and into multiple fields.

Example 1: map keys to text

```
</DTMF>
<reco id="speech" >
  <grammar src="/grm/1033/digits.xml" />
  <bind targetElement="credit_card_number" />
</reco>
```

4.2 Attributes and properties

4.2.1 Attributes

- **dtmfgrammar:** Required. The URI of a DTMF grammar.

4.2.2 Properties

- **DTMFgrammar** Read-Write.

An XML DOM Node object representing DTMF to string conversion matrix (also called DTMF grammar). The default grammar is

```
<dtmfgrammar>
  <key value="0">0</key>
  <key value="1">1</key>
  ...
  <key value="9">9</key>
  <key value="+">*</key>
  <key value="#">#</key>
</dtmfgrammar >
```

flush

Read-Write, a Boolean flag indicating whether to automatically flush the DTMF buffer on the underlying telephony interface card before activation. Default is false to enable type-ahead.

escape

Read-Write. The escape key to end the DTMF reading session. Escape key is one key.

- **numDigits**
Read-Write. Number of key strokes to end the DTMF reading session. If both escape and length are specified, the DTMF session is ended when either condition is met.
- **dtmfResult**
Read-only string, storing the DTMF keys user has entered. Escape is included in result if -typed.
- **text**
Read-only string storing white space separated token string, where each token is converted according to DTMF grammar.
- **initialTimeout**
Read-Write. Timeout period for receiving the first DTMF keystroke, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.
- **interdigittimeout**
Read-Write. Timeout period for adjacent DTMF keystrokes, in milliseconds. If unspecified, defaults to the telephony platform's internal setting.

4.3 Object methods:

4.3.1 Start

Enable DTMF interruption and start a DTMF reading session.

Syntax:

```
Object.Start( );
```

Return value:
None

Exception:
None

4.3.2 Stop
Disable DTMF. The key strokes entered by the user, however, remain in the buffer.

Syntax:

```
Object.Stop( );
```

Return value:
None

Exception:
None

4.3.3 Flush

Flush the DTMF buffer. Flush can not be called during a DTMF session.

Syntax:

```
Object.Flush( );
```

Return value:
None

Exception:
None

4.4 Events

4.4.1 onKeyPress

Fires when a DTMF key is press. This overrides the default event inherited from the HTML control. When user hits the escape key, the onRec event fires, not onKeyPress.

Syntax:

<pre><DTMF onKeyPress="handler"></pre>	Inline HTML
<pre>Event property</pre>	Event property - handler Object.onKeyPress = obj.onKeyPress = GetRef("handler");

Event Object Info:

Bubbles No	To invoke Press on the touch-tone telephone key pad
Default action Returns the key being pressed	

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.2 onReco

Fires when a DTMF session is ended. The event disables the current DTMF object automatically.

Syntax:

Inline HTML	<DTMF onRecog="handler" ...>
Event property	Object.onRecog = handler Object.onRecog = GetRef("handler");

Event Object Info:

Bubbles	No
To invoke	User presses the escape key or the number of key strokes meets specified value.
Default	Returns the key being pressed action

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

4.4.3 ontimeout

Fires when no phrase finish event is received before line out. The event halts the recognition process automatically.

Syntax:

Bubbles	No
To invoke	No DTMF key stroke is detected within the timeout specified.
Default	None

Event Properties:

Although the event handler does not receive properties directly, the handler can query the event object for data.

5 callControl Object

Represents the telephone interface (call, terminal, and connection) of the telephone voice browser. This object is as native as window object in a GUI browser. As such, the lifetime of the telephone object is the same as the browser instance itself. A voice browser for telephony instantiates the telephone object, one for each call. Users don't instantiate or dispose the object.

At this point, only features related to first-party call controls are exposed through this object.

5.1 Properties**Syntax:**

Inline HTML	<DTMF onTimeout="handler" ...>
Event property (in ECMAScript)	Object.onTimeout = handler Object.onTimeout = GetRef("handler");

Event Object Info:

- address**
Read-only. XML DCM node object. Implementation specific. This is the address of the caller. For PSTN, may a combination of NNI and ALI. For VoIP, this is the caller's IP address.

(132) 特別2003-131772

- **ringsBeforeAnswer**
Number of rings before answering an incoming call.
Default is infinite, meaning the developer must specifically use the `Answer()` method below to answer the phone call. When the call center uses ACD to queue up the incoming phone calls, this number can be set to 0.

5.2 Methods

Note: all the methods here are synchronous.

5.2.1 Transfer

Transfers the call. For a blind transfer, the system may terminate the original call and free system resources once the transfer completes.

Syntax:

```
telephone.Transfer(strText);
```

Parameters:

- strText: Required. The address of the intended receiver.

Return value:

None.

Exception:

Throws an exception when the call transfer fails, e.g., when end party is busy, no such number, fax or answering machine answers.

5.2.2 Bridge

Third party transfer. After the call is transferred, the browser may release resources allocated for the

(133) 特別2003-131772

call. It is up to the application to recover the session state when the transferred call returns using strUID. The underlying telephony platform may route the returning call to a different browser. The call can return only when the recipient terminates the call.

Syntax:

```
telephone.Bridge(strText, strUID, [imaxTime]);
```

Parameters:

- strText: Required. The address of the intended receiver.
- strUID: Required. The session ID uniquely identifying the current call. When the transferred call is routed back, the strUID will appear in the address attribute.
- imaxTime: Optional. Maximum duration in seconds of the transferred call. If unspecified, defaults to platform-internal value.

Return value:

None.

Exception:

None.

5.2.3 Answer
Answers the phone call.

Syntax:

```
telephone.Answer();
```

Return value:

None.

Exception:
Throws an exception when there is no connection.
No onAnswer event will be fired in this case.

5.2.4 Hangup
Terminates the phone call. Has no effect if no call currently in progress.

Syntax:
`telephone.Hangup();`

Return value:

None.

Exception:

None.

5.2.5 Connect
Starts a first-party outbound phone call.

Syntax:
`telephone.Connect(strText, [iTimeout]);`

Parameters:

- **strText:** Required. The address of the intended receiver.
- **iTimeout:** Optional. The time in milliseconds before abandoning the attempt. If unspecified, defaults to platform-internal value.

Return value:

None.

Exception:

Throws an exception when the recording can not be written to the url.

5.3 Event Handlers

App developers using telephone voice browser may implement the following event handlers.

Exception:
Throws an exception when the call cannot be completed, including encountering busy signals or reaching a FAX or answering machine (Note: hardware may not support this feature).

5.2.6 Record
Record user audio to file.

Syntax:
`telephone.Record(url, endSilence, [maxTimeout], [initialTimeout]);`

Parameters:

- **url:** Required. The url of the recorded results.
- **endSilence:** Required. Time in milliseconds to stop recording after silence is detected.
- **maxTimeout:** Optional. The maximum time in seconds for the recording. Default is platform-specific.
- **initialTimeout:** Optional. Maximum time (in milliseconds) of silence allowed at the beginning of a recording.

Return value:

None.

Exception:

Throws an exception when the recording can not be written to the url.

```

特蘭 2003-131772          特蘭 2003-131772

(136)          (137)

5.3.1 onIncoming( )          p_pin.Start(); q_login.Start(); dtmf.Start();

Called when the voice browser receives an incoming
phone call. All developers can use this handler to read
caller's address and invoke customized features before
answering the phone call.

5.3.2 onAnswer( )
Called when the voice browser answers an incoming phone
call.

5.3.3 onHangup( )
Called when user hangs up the phone. This event is NOT
automatically fired when the program calls the Hangup
or Transfer methods.

5.4 Example
This example shows scripting wired to the call control
events to manipulate the telephony session.

<HTML>
<HEAD>
<TITLE>Logon Page</TITLE>
<SCRIPT>
var focus;
function RunSpeech() {
    if (logon.user.value == "") {
        focus="user";
        P_UID.Start(); q_login.Start(); dtmf.Start();
        return;
    }
    if (logon.pass.value == "") {
        focus="pin";
    }
}
</SCRIPT>
<BODY>
<reco id="g_login"
onReco="login_reco(); runSpeech()">

```

```

時間 2003-131772
(138)

<timecu="5000"
onTimeout="p_miss.start(); RunSpeech() ;>
grammar
src="http://kokoneel/atzadedemo/speechconomy/login.html"/>
<dtnf id="dtnf">
</reco>
<prompt id="p_main">Please say your user ID and pin
number</prompt>
<prompt id="p_uid">Please just say your user I D</prompt>
<prompt id="p_pin">Please just say your pin number</prompt>
<prompt id="p_miss">Sorry, I missed that</prompt>
<prompt id="p_thank">Thank you. Please wait while I verify
Your identity</prompt>
<prompt id="p_retry">Sorry, your user ID and pin number do
not match</prompt>
<H2>Login</H2>
<form id="logon">
  <input type="text" name="user" onChange="runSpeech() ;>
  <input type="password" name="pass" onChange="RunSpeech() ;>
</form>

```

```

(139) 粤通 2003-131772

"tooltip" mechanism) to help form the content of the help
prompt.

<html>
  <title>Context Sensitive Help</title>
  <head>
    <script>
      var focus;
      function RunSpeech() {
        if (trade_stock.value == "") {
          focus="trade_stock";
          p_stock.start();
          return;
        }
        if (trade_op.value == "") {
          focus="trade_op";
          p_op.start();
          return;
        }
        //.. repeat above for all fields
        trade.submit();
      }
      function handle() {
        res = event.srcElement.result;
        if (res.text == "help") {
          text = "Please just say";
          text += document.all[focus].title;
          p_help.Start(text);
        } else {
          // proceed with value assignments
        }
      }
    </script>
  </head>
  <body>
    <prompt id="P_help" oncomplete="checkFields()" />
    <prompt id="P_stock" oncomplete="g_stock.Start()" />
    Please say
    the stock name</prompt>
    <prompt id="P_op" oncomplete="g_op.Start()" />
    Do you want to
    buy or sell</prompt>
    <prompt id="P_quantity" oncomplete="g_quantity.Start()" />
    How
    many shares?</prompt>
    <prompt id="P_price" oncomplete="g_price.Start()" />
    What's the
    price?</prompt>

```

特開2003-131772

(141)

```

<reco id="g_stock" onReco="handle(); checkFields()" />
  <grammar src=". /g_stock.xml" />
    </ reco >
<reco id="g_qty" onReco="handle(); checkFields()" />
  <grammar src=". /g_qty.xml" />
    </ reco >
<reco id="g_quantity" onReco="handle(); checkFields()" />
  <grammar src=". /g_quant.xml" />
    </ reco >
<reco id="g_price" onReco="handle(); checkFields()" />
  <grammar src=". /g_price.xml" />
    </ reco >

```

```
<form id="trade">
```

```
<input name="stock" title="stock name" />
    <select name="op" title="buy or sell">
        <option value="buy" />
        <option value="sell" />
    </select>
    <input name="quantity" title="number of shares" />
    <input name="price" title="price" />
</form>
```

Ergonomics

The following example shows activation of prompt and record elements using SMIL mechanisms.

```

<html xmlns:t="urn:schemas-microsoft-com:time"
      xmlns:sp="urn:schemas-microsoft-com:speech">
  <head>
    <style>
      .time {
        behavior: url (#default#time2);
      }
    </style>
  </head>
  <body>
    <input name="txtBoxOrigin" type="text"/>
    <input name="txtBoxDest" type="text" />
  </body>

```

特開2003-131772

特開2003-131772

```

<sp:prompt class="time" :begin="0">
    Please say the origin and destination cities
</sp:prompt>
<t:par t:begin="time.end" t:repeatCount="indefinitely">
    .<sp:reco Class="time" >
        <grammar src=". /city.xml" />
        <bind target-Element="txtBoxOrigin"
            value=". /origin_city" />
        <bind target-Element="txtBoxDest"
            test=". /snl /dest_city [econ_dence $gt; 40]"
            value=". /des_.city" />
    </sp:reco>
</t:par>

```

7 SMEX (Message) element/object

SMEX, short for Simple Messaging EXchange/Extension, is an object that communicates with an external component or application on the platform of the client device. It can be embedded into an XML or similar markup based document as an element with the tag name <smex>. Exemplary usages of the messaging object may include logging and telephony controls. The object represents the extensibility of markup based recognition and prompting as it allows new functionality be

Upon instantiation, the object is directed to establish an asynchronous message exchange channel with a platform component or application through its configuration parameters or attribute specifications. It has a string property whose content is sent to the platform component or application whenever the property is the recipient of an

(142) 特開2003-131772

assignment operation (i.e., `lvalue`). Similarly, it also has a property of XML DOM Node type that holds the message received from the platform component or application. The message object sends an event whenever it receives a Platform message. Since its basic operations are asynchronous, the object also has a built-in clock for the application developers to manipulate timeout settings.

The message or smex object is agnostic to the means of communications. However, in one embodiment, the smex object has the same life span as ordinary XML or markup elements, namely, the smex object will be destroyed when its hosting document is unloaded. While in many cases, the smex object can perform automatic clean-up and release communication resources when it is unloaded, there might be use cases (e.g., call controls) in which a persistent communication link is desirable across markup pages. For those cases, the architecture places the responsibility of relinquishing the allocated resources (e.g. close the socket) on the application developers.

The smex object is neutral on the format (schema) of messages. In some embodiments, it may desirable to require implementers to support a few rudimentary schemas, with strong preferences to existing standard messages formats (e.g. as used in SIP or CCXML). In essence, the architecture allows both the platform and application developers to take the full advantage of the standardized extensibility of XML or similar markup to introduce other features without losing interoperability in the mean time.

Example 1: use of smex as a logging object

(143) 特開2003-131772

```
<smex id="logServer">
  <Param name="ds:server" xmlns:d="urn:Microsoft.com/COM">
    <d:protocol>IX:OH</d:protocol>
    <d:clsid>20930930293020930942098432098</d:clsid>
    <d:iid>0903859304903498530985309094503</d:iid>
  </Param>
</smex>
```

```
<listen ...>
  ...// other directives binding reco results to input
  fields
    <bind targetElement="logServer" targetAttribute="sent"
      value="*[@log $geq 3]"/>
  </listen>
```

This example demonstrates how a logging mechanism can be achieved using a COM object with its class id and interface id. The speech developers attach an attribute "log" indicating the level of interests for logging to the relevant SML nodes. In the example above, the app developer chooses to log all nodes with log value greater or equal to 3 by using a single bind directive. The example works in both downlevel and uplevel browsers.

The example also intends to demonstrate it is possible for a page to contain multiple smex objects communicated with the same platform component as long as there won't be confusion on which smex object is responsible for delivering the platform messages back to the recognition document. The above example implies a component can implement multiple interfaces, each of which has its own smex or message

Conduit. The same argument applies to TCP servers listening to multiple ports.

Example 2: Reading the addresses for an incoming call:

```
    ...  
    <smex id="telephone" sent="start_listening">  
        <param name="server">http://tel-svr/whatever</param>  
        <bind targetElement="session_id" value="//sid"/>  
        <bind targetElement="remote" value="//remote_addr"/>  
        <bind targetElement="transfer"  
            value="//transfer_addr"/>  
        <bind targetElement="local" value="//local_addr"/>
```

This example shows how the bind directives can be used to process the received message. The example assumes a message for incoming call to have sub-elements `remote_addr`, `transfer_addr`, and `local_addr` whose contents represent the `remote`, `transfer`, and local addresses of the incoming call, respectively.

In this example, a connectionless protocol based on HMTT's used to communicate with the telephony server. The telephony server here is designed to communicate with more than one browser instances, and hence each client must identify itself with a unique id assigned by the server when the app

This is achieved in this example by sending the server a "start_listening" message. In this example, the session id is stored in a hidden field that can be sent back to the Web server and passed on to the next page of the application, although other techniques (e.g. client side cookie) can also be used to manage session state. As is in the case for `reco`, not every bind directives will be executed for every platform message. The above example does not imply the unique id will only be received when there is an incoming phone call.

Properties 71

- The smex object can have the following properties, with only the Read/write properties allowed to also serve as attributes for the initial value specification.
 - **sent:** Read/Write, a string corresponding to the message to be sent to the platform component. Whenever sent is used as a lvalue, its contents are dispatched. There is no effect when the property is used as a rvalue or when a null object is assigned to this property.
 - **received:** Read Only, an XML DOM Node data indicating the received message. The message will be available as a rvalue until the next onreceive event is ready to send.
 - **timer:** Read/Write, a number in millisecond indicating the time span before a timerout event will be triggered. The clock starts ticking when the property is assigned a positive value. The value can be changed when a count down is in progress. A zero or negative value stops the clock without triggering the timeout event. The default is 0, meaning no timeout.

(146) 特開2003-131772

- **status:** Read Only, an integer indicating the recent status of the object. The possible values are 0, -1, and -2, which means normal, timeout expired, and communication with the platform cannot be established or has been interrupted, respectively. Platform specific error messages should be conveyed through the received property. For the cases that the error message is successfully delivered, the status code is 0.

7.2 Events

The object has the following events:

- **onReceive:** This event is sent when a platform message has arrived. If there are any directives declared by the bind elements, those directives will first be evaluated before the event is fired. Prior to the sending, the received property will be updated.
- **onError:** This event is sent when the timeout expires, or a communication link error has been encountered. When the event is sent, the status property will be updated with a corresponding error code as described above.

7.3 Child Elements

When assuming an element form, smex may have the following child elements:

- **bind:** same as in reco, except the directives are operated on the received message.
- **param:** same as in reco, provides platform specific parameters for the smex object. Each param element may be named using a "name" attribute, with the contents of the param element being the value of the parameter. In

(147)

特開2003-131772

- one embodiment, the element should understand standard XML attributes for name space and XML data type declaration.

7.4 Other Comments

One elegant way to extend SMEX for the logging function would be

```
<smex id="logServer" ...> ...
<script> function logMessage(logClass, message) {
    logServer.sent = logClass + "|" + message;
} </script>
```

This in effect extends the object with a (global) function whose behavior can be customized. In the above example, the logging function is programmed to insert a field separator "|" between the id and the message.

For those who do not like global functions, they can use the __prototype__ property of ECMAScript to attach the function as an object method. For example

```
<smex id="logServer" onload="addFunction()"> ...
<script>
function my_logMessage(logClass, message) {
    logServer.sent = logClass + "|" + message;
}
function addFunction() {
    ...
}
```

```

logServer.prototype.logMessage =
{
    my_logMessage;
}

</script>

```

One can refer to the function in a more object-oriented manner:

```

logServer.logMessage(RECO_LOG_ERROR, "My
message");

```

It is noted that more work is required from the smex object implementers in order to make the extension work as in the above example, although all the necessary mechanisms are already well established standards.

4. Brief description of the Drawings

FIG. 1 is a plan view of a first embodiment of a computing device operating environment.

FIG. 2 is a block diagram of the computing device of FIG. 1.

FIG. 3 is a plan view of a telephone.

FIG. 4 is a block diagram of a general purpose computer.

FIG. 5 is a block diagram of an architecture for a client/server system.

FIG. 6 is a display for obtaining credit card information.

FIG. 7 is a page of mark-up language executable on a client.

FIG. 8 is an exemplary page of mark-up language executable on a client having a display and voice recognition capabilities.

FIGS. 9A and 9B are an exemplary page of mark-up language executable on a client with audible rendering only and system initiative.

FIG. 10A and 10B are an exemplary page of mark-up language executable on a client with audible rendering only and mixed initiative.

FIG. 11 is an exemplary script executable by a server side plug-in module.

FIG. 12 is a pictorial illustration of a first operational mode of a recognition server.

FIG. 13 is a pictorial illustration of a second operational mode of the recognition server.

FIG. 14 is a pictorial illustration of a third operational mode of the recognition server.

FIGS. 15A and 15B are an exemplary page of declarative mark-up language executable on a client without scripting.

(150)

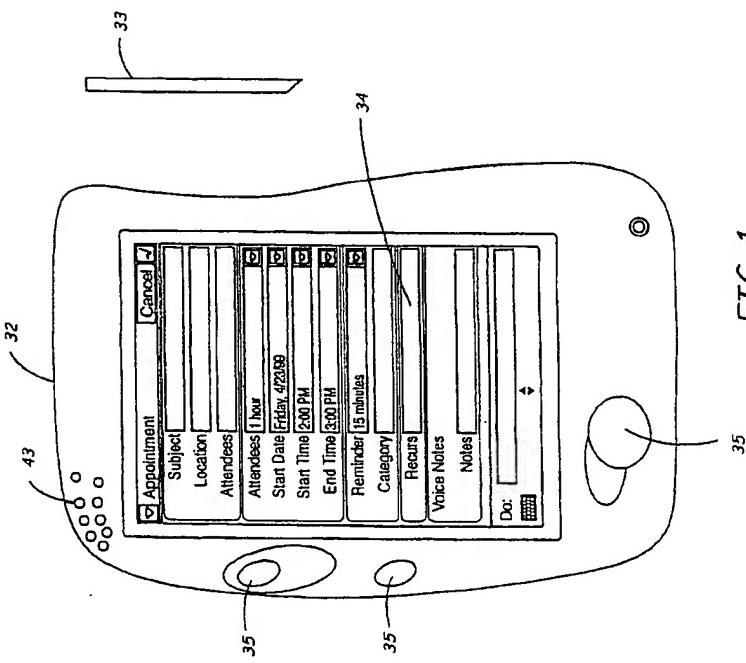


FIG. 1

(151)

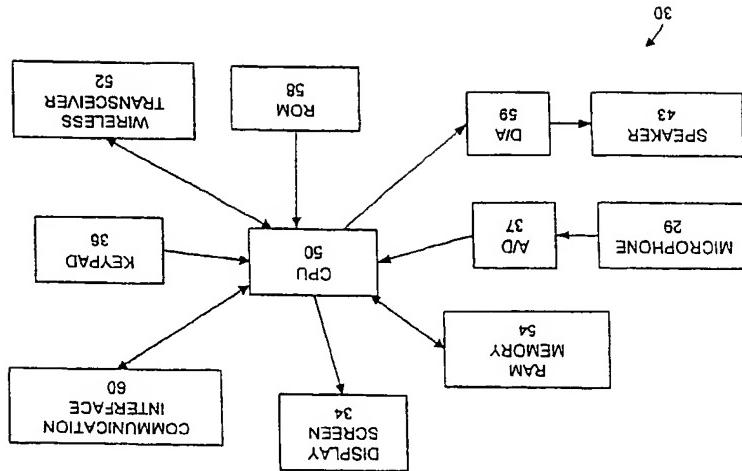
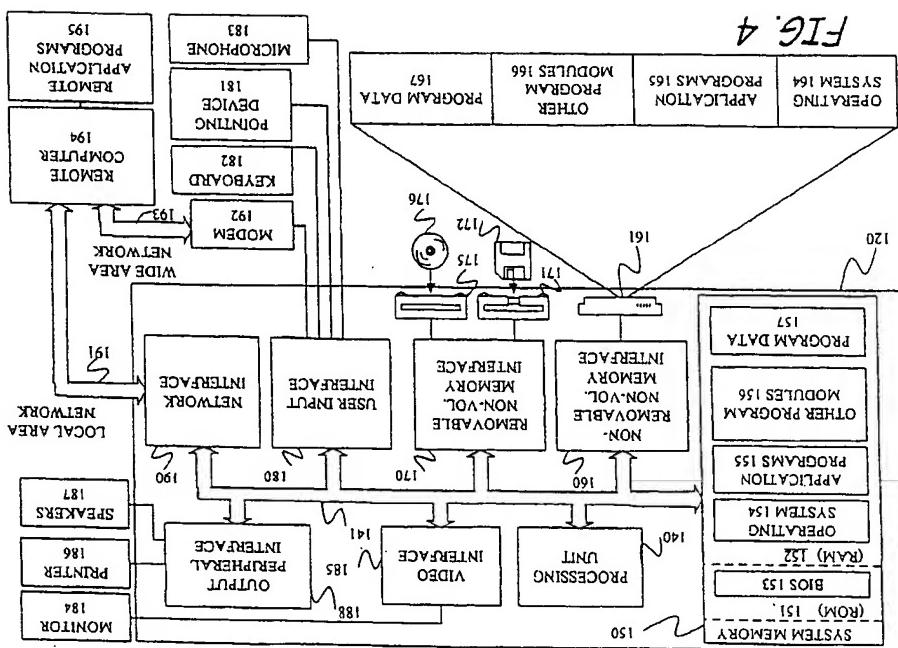
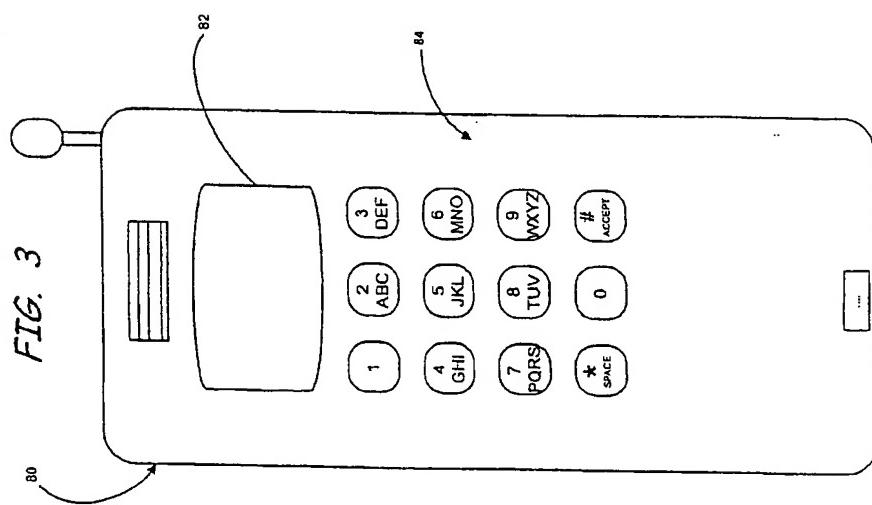


FIG. 2



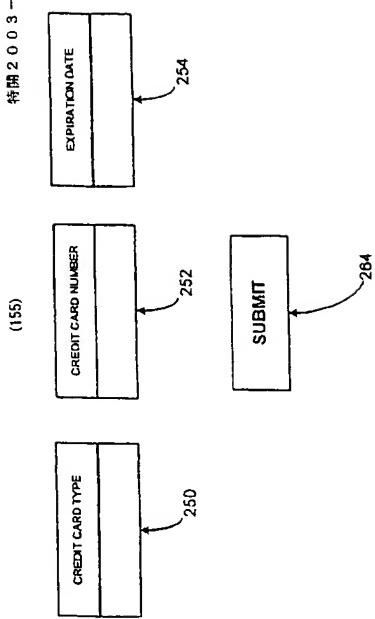


FIG. 6

FIG. 7

```

<html>
<head>
<title>Get-Card-Info, Method = GET, URL = https://www.meritxpress.com</title>
<script>
function getCardHandle() {
    let cardType = document.querySelector('input[name="card-type"]')
    let cardNumber = document.querySelector('input[name="card-number"]')
    let expirationDate = document.querySelector('input[name="expiration-date"]')
    let cvv = document.querySelector('input[name="cvv"]')

    let cardData = {
        type: cardType.value,
        number: cardNumber.value,
        expirationDate: expirationDate.value,
        cvv: cvv.value
    }

    let xhr = new XMLHttpRequest()
    xhr.open('GET', `https://www.meritxpress.com/api/get-card-info?${Object.keys(cardData).map(key => `${key}=${cardData[key]}`).join('&')}`)

    xhr.onload = () => {
        if (xhr.status === 200) {
            let response = JSON.parse(xhr.responseText)
            if (response.error) {
                alert(response.error)
            } else {
                let cardInfo = response.data
                let cardElement = document.createElement('div')
                cardElement.innerHTML = `
                    <p>Card Type: ${cardInfo.type}</p>
                    <p>Card Number: ${cardInfo.number}</p>
                    <p>Expiry Date: ${cardInfo.expirationDate}</p>
                    <p>CVV: ${cardInfo.cvv}</p>
                `
                document.body.appendChild(cardElement)
            }
        }
    }
}
</script>
</head>
<body>
<form onsubmit="getCardHandle(); return false;">
    <label>Card Type:</label>
    <select name="card-type">
        <option value="Visa">Visa</option>
        <option value="MasterCard">MasterCard</option>
        <option value="American Express">American Express</option>
        <option value="Discover">Discover</option>
        <option value="Other">Other</option>
    </select>
    <label>Card Number:</label>
    <input type="text" name="card-number" width="30" />
    <label>Expiration Date:</label>
    <input type="text" name="expiration-date" width="10" />
    <label>CVV:</label>
    <input type="text" name="cvv" width="4" />
    <br>
    <input type="button" value="Get Card Info" />
</form>
</body>
</html>

```

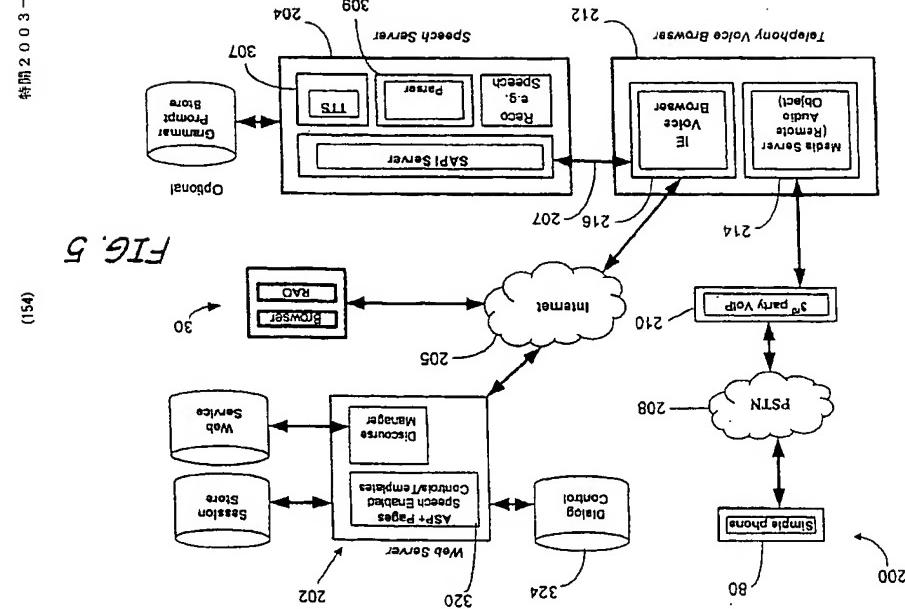


Fig. 5

```

(156)          特開2003-131772

<html>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <select name="card_type" onclick="talk(0,card_types)">
      <option value="american_express">American Express</option>
      <option value="visa">Visa</option>
      <option value="mastercard">MasterCard</option>
    </select>
    <rec id="g_card_type" onRec0="handle()">
      <grammar src="#/gram/card_types" />
    <rec0>
      <input type="text" name="card_num" width="30" />
      <onlick="talk(g_card_num, onRec0 = handle())">
        <rec id="g_card_num" onRec0 = "handle()">
          <grammar src="#" />
        </rec>
      <input type="text" name="expiry_date" />
      <onlick="talk(g_expiry_date)">
        <rec id="g_expiry_date" />
        <grammar src="#" />
      </rec>
    </rec0>
    <input type="submit" value="Submit" onclick="verify()"/>
  </form>
  <script>[(CDATA[
    function talk(gobj) {
      gobj.activate();
    }
    function handle() {
      if (get_card_info.card_num == null) {
        if (get_card_info.card_type.value == "amex") {
          if (get_card_info.card_num.length != 15)
            alert("ameX should have 15 digits");
        } else
          if (get_card_info.card_num.length != 16)
            alert("Visa and master should have 16 digits");
      }
    }
    function verify() {
      var flag = window.confirm("submit the credit card
      info?");
      if (flag) {
        get_card_info.submit();
      }
    }
  ])>
</script>
</html>

```

270 280 282 284 290 302

```

<body>
  <form id="get_card_info" method="post" action="http://payment.asp">
    <input type="button" value="welcome()" />
    <prompt id="p_welcome"> We new need your credit card </prompt>
    <prompt id="p_card_type"> I didn't understand you </prompt>
    <prompt id="p_number"> What credit card would you
    use? </prompt>
    <prompt id="p_card_num" bargain="true"> Please say the number</prompt>
    <prompt id="p_expiry_date" bargain="true"> What is the expiration
    date? </prompt>
    <prompt id="p_content">
      I have your <value select="card_type" /> <value select="card_num" />
      with expiration date
      <value select="expiry_date" />
    </prompt>
    <prompt id="p_confirm"> Is this correct? </prompt>
    <rec id="g_card_type" onRec0="onHandle(this, 2)">
      <grammar src="#" />
      <onRec0="g_card_type" onHandle="onHandle(this, 2)">
        <rec id="g_card_type" onRec0="onHandle(this, 2)">
          <grammar src="#" />
        </rec>
      <rec id="g_cardnum" onRec0="onHandle(this, 1)">
        <onRec0="handle()" onHandle="onHandle(this, 1)">
        <grammar src="#" />
      <rec id="g_expiry_date" onRec0="onHandle(this, 3)">
        <onRec0="g_expiry_date" onHandle="onHandle(this, 3)">
        <grammar src="#" />
      </rec>
    </rec0>
    <input type="text" name="card_num" value="0000000000000000" />
    <input type="text" name="expiry_date" value="00/00" />
  </form>

```

300 303 305

FIG. 8

FIG. 9A

TO FIG. 9B

FIG. 9A

```

特開2003-131772

(158)

function _handleO {
    handleO();
    checkO();
}

function checkO() {
    if (card.type.value == "") {
        p.card.type.activateO();
        g.card_type.activateO();
        return;
    }
    if (card.name.value == "") {
        p.card.name.activateO();
        g.card_name.activateO();
        vateO();
        return;
    }
    if (expire.date.value == "") {
        g.expiry_date.activateO();
        g.expiry_date.activateO();
        return;
    }
    oContent.activateO();
    p.confirm.activateO();
    confirmation.activateO();
}

function confirmed(gobj) {
    if (gobj.value == "yes") {
        getCardInfo();
    }
}

function getCardInfo() {
    // user codes start here
    // user codes start here
    if (getCardInfo.card_type.value == "Master") {
        if (getCardInfo.card_num.length != 15) {
            alert("Card number should have 15 digits");
            getCardInfo.card_num = "";
        }
        speak("Master");
    } else {
        if (getCardInfo.card_num.length != 16) {
            alert("Card number should have 16 digits");
            getCardInfo.card_num = "";
        }
        speak("Visa and MasterCard should have 16 digits");
    }
}

```

FIG. 9B

ETG 10A

特附2003-131772
(160)

特開2003-131772

```

function checkFilled() {
    if(card.type.value == "" || do_field.activateO) return;
    if(card.type.value == "") { do_field.activateO; return; }
    if(card.num.value == "") { do_field.activateO; return; }
    if(card.expiry.value == "") { do_field.activateO; return; }
    if(expiry_date.value == "") { do_field.activateO; return; }
    if(expiry_data.value == "") { do_field.activateO; return; }
    if(content.activateO); do_field.activateO;
    if(confirm.activateO); confirmation.activateO;
}

function confirmed(obj) {
    if(gob_.recognes_rect.get("yes")) {
        get_card_info.card_num = susan_.personal();
    }
}

// user codes start here

function handleO() {
    if(card == get_card_info.card_num) {
        if(card.length == 15) {
            prompt.speak("Name should have 15 digits");
            get_card_info.card_num = "";
        }
        else {
            prompt.speak("Card number should have 15 digits");
            get_card_info.card_num = "";
        }
    }
}

function gen10() {
    str = <card>credit_card type="i" number=">;
    str += card.type.value; str += "</number>";
    str += card.number.value; str += "<expire>";
    str += expiry_date.value; str += "<expire>";
    str += expiry_data.value; str += "<expire>";
    str += expire_.card_num + "</card>";
}

```

FIG. 10B

(161)

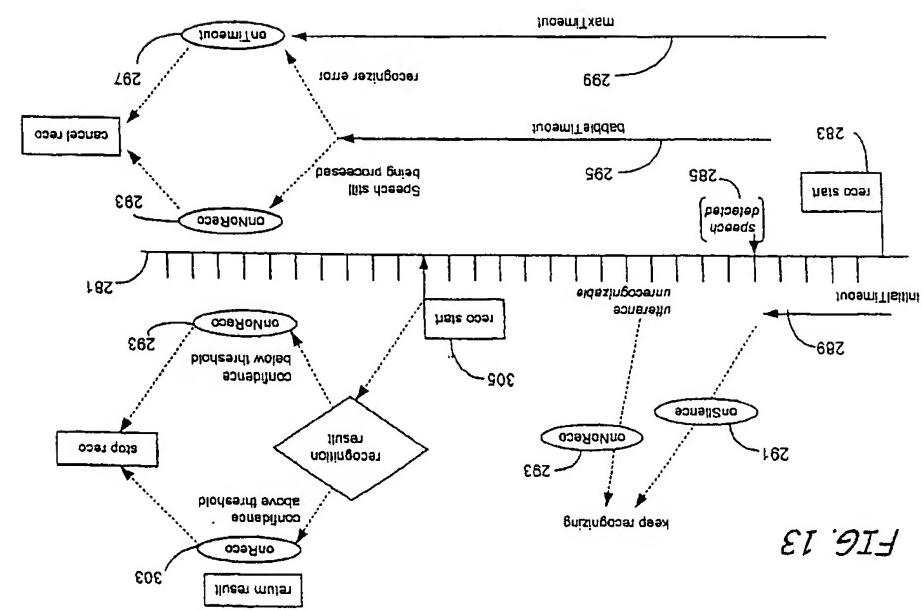


FIG. 13

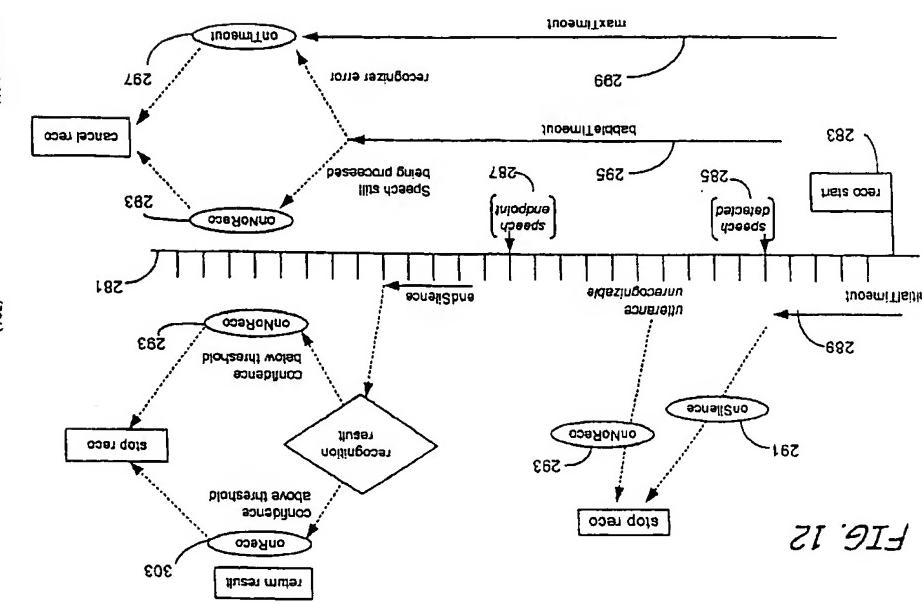


FIG. 12

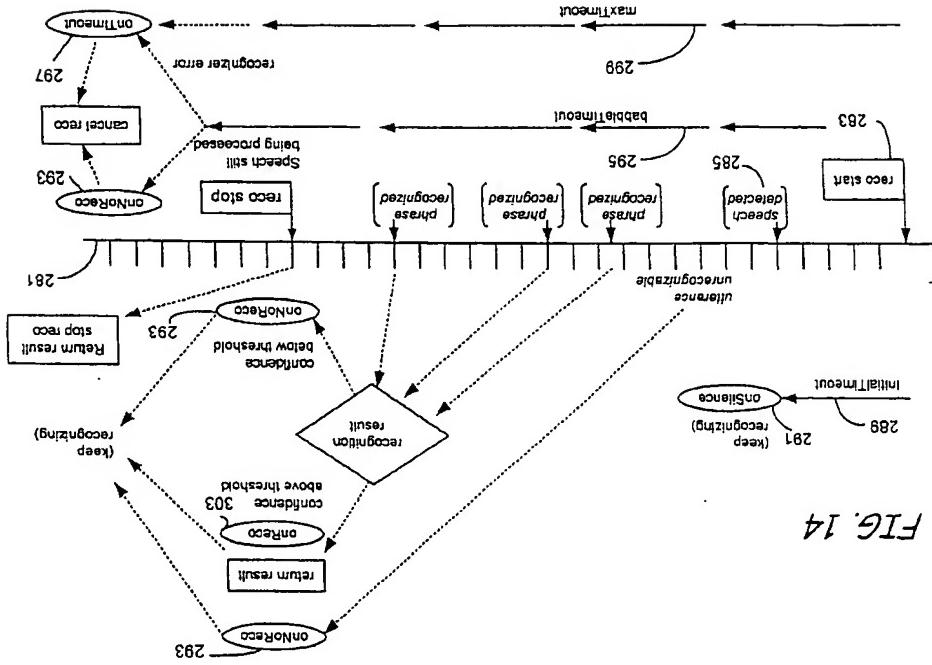


FIG. 14

特開2003-131772

特刊2003-131772

特開2003-131772

特刊2003-131772

```

<html>
  <body>
    <!-- the data section -->
    <form id="get_drink">
      <input name="drink" type="radio" value="coffee" />
      <input type="radio" name="sugar" type="hidden" />
    </form>

    <!-- The speech section -->
    <prompt id="confirm">Welcome, caller! </prompt>
    <prompt id="ask"> Do you want coke, coffee, or orange juice? </prompt>
    <prompt id="confirm"> I heard <value href="#drink"/>. Is this correct? </prompt>
    <prompt id="thanks"> Thank you. Please wait while I get it for you. </prompt>
    <prompt id="retry"> Okay, let's do this again </prompt>
    <prompt id="prompt1"> Sorry, I missed that. </prompt>
    <prompt id="sugar"> Do you want cream or sugar with your coffee? </prompt>

    <!-- The grammar section -->
    <grammar src="#drinkTypes" />
    <rule start="start">
      <bind test="@confidence $!s 10!" targetElement="reprompt" targetMethod="start" />
      <rule start="start" targetElement="reco_drink" targetMethod="start" />
      <bind test="drink/coffee (@confidence: $!s 10)" targetElement="drink" value="drink" targetMethod="start" />
      <rule start="start" targetElement="reco_sugar" targetMethod="start" />
      <bind test="cream_sugar (@confidence: $!s 10)" targetElement="reco_cream_sugar" targetMethod="start" />
    </rule>
    <rule start="start">
      <bind test="@confidence $!s 10!" targetElement="drink" value="drink" targetMethod="start" />
      <rule start="start" targetElement="reco_drink" targetMethod="start" />
      <bind test="coffee (@confidence: $!s 10)" targetElement="drink" value="drink" targetMethod="start" />
    </rule>
  </body>

```

FIG. 15A

(166) 特開2003-131772

```
<reco id="reco_cream_sugar"><grammar src="/cream+sugar"/>
  <bind test="/(confidence_sigs_10_and_
    host() /getDrink/drink = 'coffee' |"
      targetElement="cream" targetAttribute="checked"
      value="/cream/drink"
      targetElement="sugar" targetAttribute="checked"
      value="/sugar/drink"
      targetElement="confirm" targetMethod="start"
      targetElement="reco_yesno" targetMethod="start"/>
</reco>

<reco id="reco_yesno"> <grammar src="/yesno"/>
  <bind test="yes[(confidence_sigs_10 |
    targetElement="thanks" targetMethod="start"
    384   — targetElement="get_drink"
    384   — targetMethod="submit"/>
    384   — target test="no OR !(@(confidence_sigs_10 |
      targetElement="fatty" targetMethod="start"
      382   — targetElement="mask" targetMethod="start"
      382   — targetElement="reco_drink"
      382   — targetMethod="start"/>
</reco>

<!--call control section -->
<smej id="telephone" sent="start_listening"><param
  server="ccxmlproc" ...></param>
  <bind targetElement="uid" value="guid"/>
  <bind test="!call_connected"
    targetElement="welcome" targetMethod="start"
    381   — targetElement="ask" targetMethod="start"
    381   — targetElement="reco_drink"
    381   — targetMethod="start"/>
</smej>
</body>
</html>
```

FIG. 15B

1. Abstract

A markup language for execution on a client device in a client/server system includes extensions for recognition.

2. Representative Drawing
FIG. 1